Treball de Final de Grau

Grau en Enginyeria en Tecnologies Industrials

Conducció semiautomàtica de vehicle tenint en compte senyals de tràfic

- **Autor:** Iñaki Pujol Carmona
- **Director:** Josep Vilaplana Pastó
- **Convocatòria:** Abril 2020

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Resum

L'objectiu d'aquest projecte serà dissenyar i construir un prototip de vehicle de conducció semiautomàtica.

Per a aconseguir aquest objectiu serà necessari realitzar un algoritme capaç de detectar semàfors que sigui precís i robust per a evitar, sobretot, falsos positius.

Primer de tot s'introduiran els conceptes de *Machine Learning*, *Deep Learning* i *Computer vision* que són els camps amb els quals es treballarà.

Després es mostrarà el material amb el qual es treballarà, les seves característiques i perquè s'utilitzen.

I finalment, la implementació que es fa de cada eina.

Paraules clau: "Speed/accuracy trade-offs for modern convolutional object detectors. "Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z, Song Y, Guadarrama S, Murphy K, CVPR 2017, vehicle automàtic, intel·ligència artificial, visió artificial, computer vision.

Índex

Índex de figures

Índex de taules

Glossari

Algorisme: És com s'anomena al procediment de càlcul seguint unes instruccions per a arribar a una solució, especificades unes dades. També és possible anomenar-lo *Algoritme*.

Api: (*Application Programming Interface*) És una interfície de programació d'aplicacions la qual permet la interacció de diferents programes.

Arduino IDE: És una llengua de programació de la tecnologia Arduino.

Bit: Unitat mínima d'informació que entén un ordinador amb 2 possibles valors (binari): 0-1, ON-OFF, apagat-encès, etc.

Bus: Sistema de transferència de dades.

Byte: 8 **bits**.

C++: Llenguatge de programació orientat a objectes creat el 1985 provinent del seu predecessor *C*.

Compilador: Transforma un programa en un llenguatge a un altre, normalment perquè l'ordinador pugui entendre el que s'ha escrit.

Computer Vision: És la branca de la ciència que estudia la manera que un ordinador comprengui o pugui extreure tanta informació com sigui possible d'una imatge. En enginyeria és la manera més semblant que s'ha pogut trobar d'apropar-se a la visió humana.

Convolució: Operador matemàtic utilitzat per a veure com canvia una funció respecte a una altra.

CPU: *Central Process Unit*. La Unitat Central de Processament és el **maquinari** que interpreta les instruccions d'un programa informàtic.

CSI *Camera Serial Interface*. És una Interfície Serie per a Càmeres que permet comunicar l'aparell òptic amb un processador.

Dataflow: És una manera de programar basada en un **graf** dirigit.

Deep Learning: És una branca del **machine learning** basat en xarxes neuronals artificials.

Escriptori virtual: És una manera de controlar un ordinador de manera remota.

Ethernet: És la manera més comuna de connectar-se a Internet mitjançant un cable.

GPIO: *General-Purpose input/output*, entrada i sortida d'informació de propòsit general.

GPU: *Graphic Processing Unit*. La Unitat de Processament Gràfic està dedicada a la generació de gràfics en un ordinador.

GUI: *Graphical User Interface* (Interfície Gràfica d'Usuari). Interfície que utilitza elements gràfics per a permetre controlar un programa informàtic a l'usuari.

Graf: Representació d'objectes connectats.

Maquinari: (*Hardware*). La part física d'un ordinador.

I^2C**:** És un **bus** de comunicació en sèrie.

Input: Entrada, normalment de dades en aquests àmbits.

LCD: (*Liquid crystal Display*) Pantalla de Cristall Líquid. Són molt utilitzades en rellotges .

Llenguatge màquina: Sistema d'instruccions binàries que permet comunicar-te amb un microprocessador.

Machine Learning: És una disciplina científica de la Intel·ligència Artificial que crea sistemes capaços d'aprendre de manera autònoma a reconèixer patrons.

On-line: En línia, que està a Internet.

Perifèric: La part externa d'un ordinador que permet la interacció un les persones con el teclat, el ratolí, la pantalla i la càmera.

Píxel: Unitat mínima d'una imatge digital, cada punt de llum d'ella.

PWM: Modulació per amplada de polsos. En aquest projecte serveix per a regular la potència entregada a les rodes.

Python: És un llenguatge de programació orientat a objectes amb una sintaxi molt clara.

Servo: *Servomotor de modelisme*, és un actuador capaç de moure's a qualsevol posició del seu rang i mantenir-se estàtic.

Programari: (*Software*). Programes d'un ordinador.

USB: *Universal Serial Bus*. Bus en Sèrie Universal és un estàndard industrial de sistema de comunicació per a ordinadors.

Wi-Fi: És una xarxa local sense fils per a connectar-se a Internet.

1 Introducció

1.1 Motivació

La tecnologia cada cop arriba més lluny i el pas fins a la següent etapa es fa més ràpidament i amb més informació que l'anterior. És per això que dóna lloc a buscar **algorismes** cada cop més eficaços i robustos per a tal d'ajudar-nos a processar la gran quantitat d'informació més ràpidament i amb menys marge d'error.

Aquest fet ens porta a la capacitat de computació d'una **CPU** perquè permet fer aquestes anàlisis de la informació proporcionada sense cap classe de descans ni baixar l'eficàcia a mesura que passa el temps. D'aquesta manera cal programar l'**algorisme** de forma robusta per a mirar de disminuir el màxim possible els falsos positius, perquè no ha de parar a descansar fa molts més anàlisis per hora i si no és robust, també farà molts errors provinents dels errors a l'hora de programar.

També és de vital importància reduir tant com sigui possible el temps en què es tarda a fer cada operació per a augmentar l'agilitat que processa informació, i la manera de fer-ho és utilitzar només la memòria justa i exacta per a cada pas. I aquí entra en joc la capacitat humana: per a poder trobar noves maneres de fer un sistema més robust i utilitzant menys memòria per a poder ser més dinàmic.

En aquest moment hi ha dos camps que s'estan guanyant molt protagonisme últimament: la conducció automàtica de vehicles i el **Deep Learning**. Els quals són àmpliament compatibles a l'hora de poder ajudar a millorar la manera i seguretat de conduir un vehicle perquè en primera instància pot servir per avisar al conductor humà d'esdeveniments que podria haver passat per alt i limitar part de les accions per a mantenir una circulació segura, i també es podria aplicar, juntament amb altres tecnologies, per a una conducció completament automàtica.

D'aquesta manera la motivació d'aquest projecte és la construcció i utilització d'un vehicle capaç de reaccionar anticipant-se a diferents senyals de tràfic que es pugui trobar en el seu camp de visió.

1.2 Objectius

1. Implementar un sistema autònom que sigui capaç de reaccionar a semàfors.

2. Programar un **algorisme** eficaç de detecció de semàfors.

    - Valorar l'eficàcia de l'**algorisme** en diferents situacions potencialment confuses.

1.3 Abast del projecte

La intenció d'aquest projecte és dissenyar i implementar un *prototip* de vehicle que pugui circular de forma autònoma, en una trajectòria rectilínia, respectant semàfors.

D'aquesta manera s'ha agafat un model de vehicle que tingui les funcions bàsiques de mobilitat i de velocitat d'un automòbil alhora que disposa d'una càmera per permetre la identificació de senyalitzacions disposades al llarg del recorregut.

Les imatges recollides per la càmera es tractaran de manera que sigui possible localitzar la informació del seu voltant i descartar la innecessària. I aquest és el punt més important del projecte: l'**algorisme** de detecció i classificació d'objectes en el camp de visió del receptor.

Així, ha sigut necessari crear prototips de semàfors que recreïn el comportament dels seus semblants als carrers de les ciutats i els senyals de tràfic corresponents.

Dit això, s'han de realitzar les següents tasques:

1. Dissenyar i implementar l'**algorisme** de detecció d'objectes.
2. Dissenyar i implementar l'**algorisme** de classificació d'objectes.
3. Dissenyar i implementar el control dels motors i servos.
4. Combinar tots els **algorismes** anteriors.
5. Dissenyar i implementar la seqüència dels semàfors.
6. Construir els semàfors.
7. Construir elements de "distracció"del sistema per a comprovar l'eficàcia i la robustesa.

2 Estudi previ

2.1 Machine Learning

El **machine learning** és una branca de la *intel·ligència artificial* que permet crear sistemes que són capaços d'aprendre patrons complexos i anar millorant-se per a predir respostes de manera autònoma.

Moltes grans empreses, com *Netflix, Spotify, Amazon, Google o Apple* utilitzen aquestes eines per a millorar la interacció individualitzada per a cada client amb els que treballen i, en els casos d'*Alexa* i *Siri* d'Amazon i Apple respectivament, per a millorar la fluïdesa i naturalitat de la veu sintetitzada que utilitzen com a mitjà d'interacció.

2.1.1 Big Data i Empresa

Hi ha un exemple bastant explicatiu a la pàgina web de cleverdata de com es pot aplicar la gran quantitat de dades de les quals disposa una empresa en el sector telefònic i aplicant el **machine learning** es podria arribar la quantitat de clients que es donaran de baixa per a intentar millorar la seva experiència i que canviïn de parer.

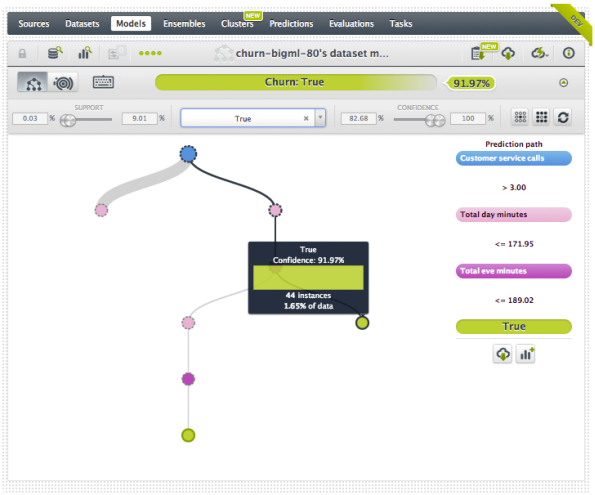
La quantitat d'informació de què disposa una empresa d'aquestes característiques de cada persona i la immensitat de clients que han tingut des de la fundació és massa gran perquè cap persona humana sigui capça d'analitzar-la i buscar qualsevol mena de patró. Per això el **machine learning** és tan important en aquests casos perquè permet, amb **algorismes** basats en l'estadística, localitzar patrons de comportament i donar un veredicte un cop introduïdes les dades.

La Figura 1a és una predicció utilitzant un sistema molt simplificat del que podria ser l'algorisme explicat anteriorment. En aquest cas el client ha realitzat:

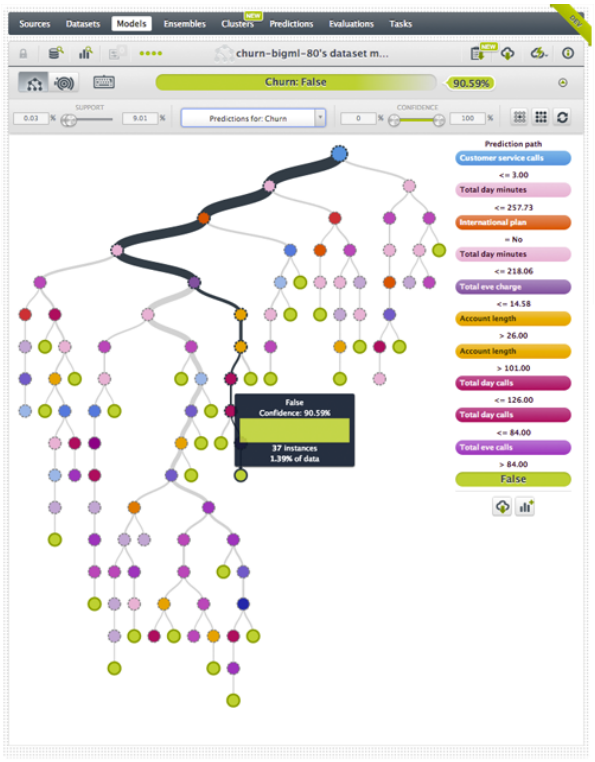
- Més de 3 trucades al servei d'atenció al client.
- Menys de 171,95 minuts al dia.
- Menys de 189,02 minuts en horari nocturn.

Segons aquest model i el client analitzat, aquest té un 91,97% de possibilitats de deixar de ser client d'aquesta empresa basant-se en el comportament de clients anteriors. Seguint amb aquest exemple, si l'empresa s'adona d'aquest fet, pot mirar de treballar per a intentar evitar que se'n vagi consultant els problemes que ha tingut amb atenció al client i mirant de millorar la seva percepció de l'empresa i, possiblement, la de molts altres.

Com podem apreciar, l'arbre complet (Figura 1b) té molts més nodes i paràmetres a tenir en compte que el simplificat i d'aquí ve la gran diferència de grandària i segurament podria arribar a donar resultats molt diferents en cas que en simplificar no s'haguessin



(a) Representació gràfica de l'algorisme reduït de predicció



(b) Arbre complet de predicció

Figura 1: Arbres de l'algorisme [Font: cleverdata (Andrés González)]

escollit correctament les característiques de més pes tal com indica l'estadística.

D'aquesta manera el **machine learning** ens pot ajudar com a empresa a ser més competitiu en el mercat laboral, fer ús de totes les dades de què es disposa actualment (ja que estem en l'era de la informació) cosa que abans era gairebé impossible i trobar patrons difícils de trobar sense aquesta ajuda.

2.1.2 Aplicacions

Actualment ja s'està utilitzant en molts camps i empreses per a millorar la qualitat dels seus productes o serveis com poden ser:

- Recomanar productes en tendes **on-line**.
- Els anuncis que apareixen en les webs.
- Fraus en transaccions.
- Predir el tràfic urbà.
- Fer diagnòstics mèdics.

2.2 Computer Vision

El **computer vision** és una branca del **machine learning** (Apartat 2.1) basat en intentar replicar part de la visió humana i la capacitat d'aquesta d'identificar objectes i classificar-los de la manera més semblant a com ho podria fer una persona a través xarxes neuronals.

Figura 2: YOLO Multi-Object Detection And Classification [Font: Towards Data Science (Ilija Mihajlovic)]

La clau del gran creixement d'aquesta disciplina en els últims temps ha sigut la capacitat d'entrenar aquests sistemes a causa de la gran quantitat d'informació i, sobretot, d'imatges que es fan cada dia al món i que es comparteixen a la xarxa (*al voltant de 3 bilions al dia*) de manera que tothom les pot veure.

Aquest fet ha permès que la taxa de precisió en la detecció d'objectes hagi passat del 50 al 99% en menys d'una dècada a l'hora que permet ser més ràpid i segur que l'ull humà.

2.2.1 Com funciona?

De la mateixa manera que el **machine learning** intenta imitar la manera de funcionar de la ment humana però realment no se sap realment com funciona aquesta, en el **computer vision** tampoc se sap com de precís és la manera que s'enfoca actualment, ja que no se sap realment com el cervell, combinat amb els ulls, processen les imatges que veiem i ens permet veure el món tal com el coneixem.

Ja que aquest camp es basa en reconèixer patrons en imatges, la manera d'entrenar un d'aquests sistemes és subministrar-li tantes imatges com sigui possible i si és possible que estiguin etiquetades per, d'aquesta manera, poder classificar.

Un cop subministrat tota aquesta informació, cal passar-la per diferents algorismes que permetin la localització de patrons que es puguin associar a cada etiqueta. És a dir que es genera una "descripció"del que significa que hi hagi cert tipus d'objecte en una imatge perquè quan al sistema se li demana que ens doni una resposta sigui capaç de dir si a la imatge hi ha alguna part que coincideixi amb la descripció que en té.

Una imatge, tal com l'entén un ordinador i com es treballarà en aquest projecte, és simplement una matriu de valors on cada valor és la informació d'un **píxel**, o part d'aquest, de la imatge.

Per a cada tipus d'imatge pot ser una mica diferent la representació d'aquesta, per exemple si la imatge és en escala de grisos, cada cel·la de la matriu té un número associat entre 0 i 255 que correspon a 1 **byte** tenint només 1 capa. El 0 representa el negre i el 255 el blanc tal com indica la figura 4.

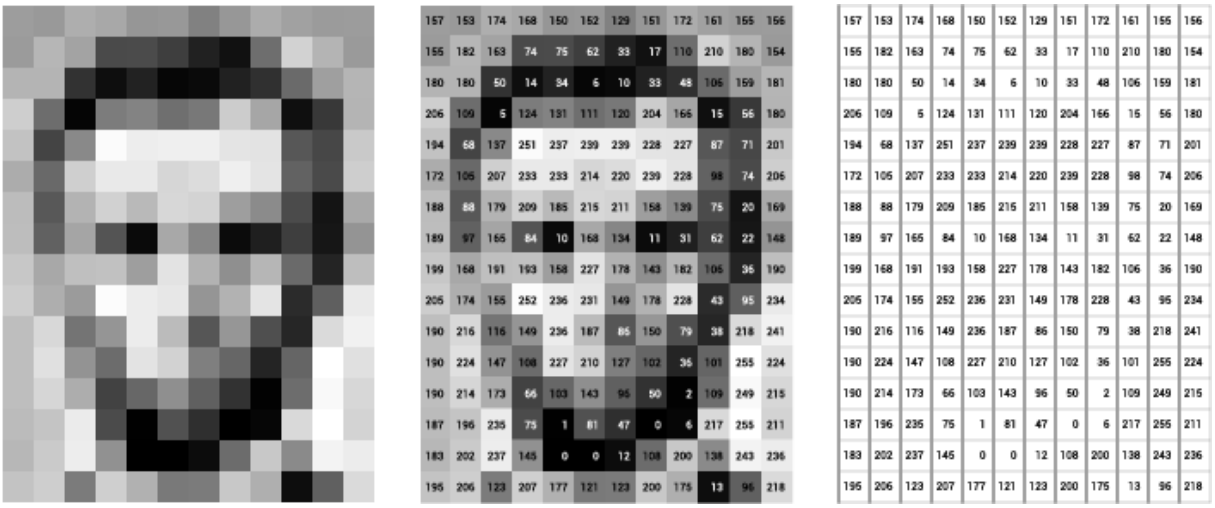


Figura 3: Esquerra: Imatge de Lincoln; Centre: Cada píxel amb el seu valor associat; Dreta: Només els valors representats en una matriu. [Font: Towards Data Science (Ilija Mihajlovic)]

En canvi, si és una imatge en color, hi ha diferents maneres de representar les matrius, una de les més comunes és en RGB (*RedGreenBlue* Vermell Verd Blau) on cada color és una capa de la imatge on pot cada cel·la de cada capa pot prendre els valors entre 0 i 255 de nou. Així aquest tipus d'imatges tenen 3 capes. També és possible afegir una capa més anomenada el canal *alpha* que representa el grau de transparència entre el mateix rang de valors que les altres capes.

Figura 4: Com funciona el RGB [Font: Presentitude a SlideShare (Presentitude)]

Quan es treballa amb **deep learning** és una pràctica molt comuna aplanar les matrius (convertir la matriu original en una unidimensional posant cada fila al final de l'anterior) i també s'acostuma a treballar amb imatges en escala de grisos perquè es redueix la quantitat d'informació requerida per a cada una d'elles. Aquesta última pràctica és deguda que, perquè l'aprenentatge sigui efectiu, cal proporcionar una gran quantitat d'imatges que ocuparan un volum considerable. La reducció a gris fa que aquest volum es redueixi a un terç, fent un ús més eficient dels recursos.

Un cop s'ha entès com és una imatge per a un ordinador i se l'ha tractat adequadament per a poder introduir-la al sistema, es pot tirar per diferents camins:

- **Segmentació d'imatge:** Separar la imatge en subimatges per a tractar-les per separat.
- **Detecció d'objectes:** Identifica un tipus d'objecte en una imatge. En els sistemes més avançats i complexos es pot trobar la **Detecció múltiple d'objectes**, la qual detecta diferents objectes en una sola imatge (com a la Figura 2)
- **Reconeixement facial:** Una aplicació molt concreta i molt utilitzada pels mòbils avui en dia com és la capacitat exclusiva de detectar un rostre facial humà identificant cada individu.
- **Detecció de límits:** Tècnica utilitzada per a detectar els límits d'un objecte.
- **Detecció de patrons:** Detecta patrons tant de forma, de color com qualsevol altre que pugui trobar.
- **Classificació d'imatges:** Agrupa les imatges d'un tipus sota etiquetes.

- **Coincidències en imatges:** Troba semblances en imatges per a ajudar a classificar-les.

Com es pot observar cada cop que es fa un sistema, s'acostumen a utilitzar diversos d'aquestes maneres, d'enfocar el problema, combinades per a arribar a un resultat òptim.

Per a crear un sistema de **computer vision** abans era necessari crear manualment una base de dades amb totes les imatges etiquetades correctament i, cada una d'elles, amb totes les característiques que definien l'objecte o patró en qüestió que mostraven. Un cop el **machine learning** va arribar als enginyers dedicats a aquest camp, molts dels problemes que abans semblaven gairebé insalvables, es van agafar des d'una perspectiva molt més favorable i segura com predir la probabilitat de supervivència del càncer de mama encara que amb moltíssim personal i temps.

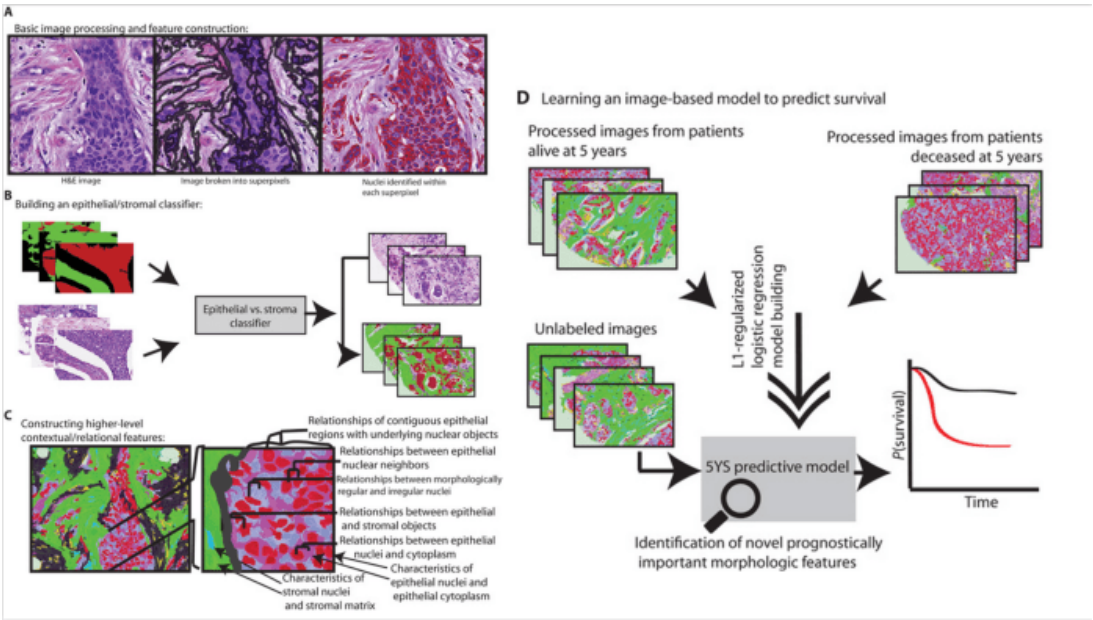


Figura 5: Exemple de machine learning amb mètodes clàssics [Font: Towards Data Science (Ilija Mihajlovic)]

Però el **deep learning** va arribar un pas més enllà, ja que treballa amb xarxes neuronals (Figura 6) amb les quals, teòricament, es pot resoldre qualsevol problema un cop el sistema s'ha nodrit amb suficients exemples, en el nostre cas amb suficients imatges. En l'apartat 2.3 es profunditza més aquest concepte.

2.2.2 Aplicacions

Hi ha molts camps en què s'aplica el **computer vision**, ja que permet dissenyar les solucions als problemes de manera més semblant a com vivim el dia a dia Però amb un

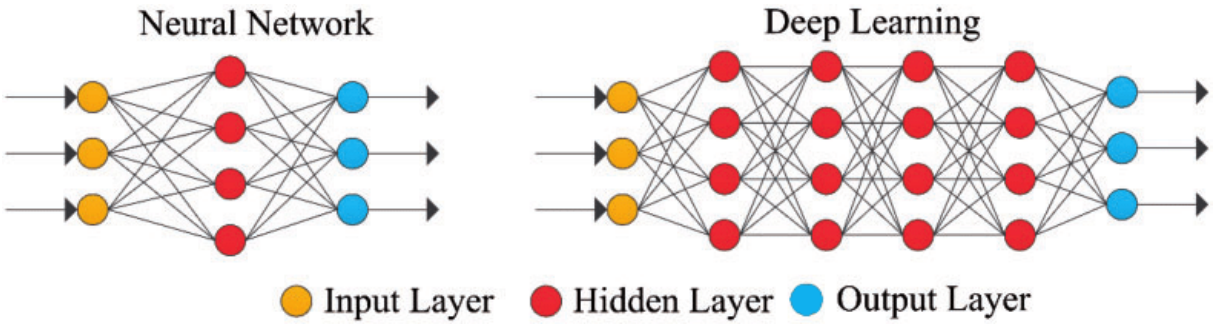


Figura 6: Exemple de la representació gràfica de la xarxa neuronal d'un sistema de Deep Learning [Font: ResearchGate]

temps de reacció molt més petit.

- **Cotxes amb conducció automàtica:** Aquest és el tema principal del projecte. Aquest tipus de cotxe porten càmeres per a cobrir diferents punts crítics per a permetre detectar els límits de la carretera i carrils, vianants, senyals de tràfic, els cotxes que l'envolten i altres possibles obstacles.
- **Reconeixement facial:** Com a mida de seguretat tal com ja s'ha comentat anteriorment.
- **Sanitat:** Permet detectar malalties com el càncer només escanejant una placa de radiografia que s'hagi fet a un pacient de forma instantània.

2.3 Deep Learning

El **deep learning** és, com hem vist a l'apartat 2.2, l'eina que ha permès realment realitzar tasques molt complexes relacionades amb el **computer vision** que abans es veien inviables a causa del material i temps requerits per a la seva realització.

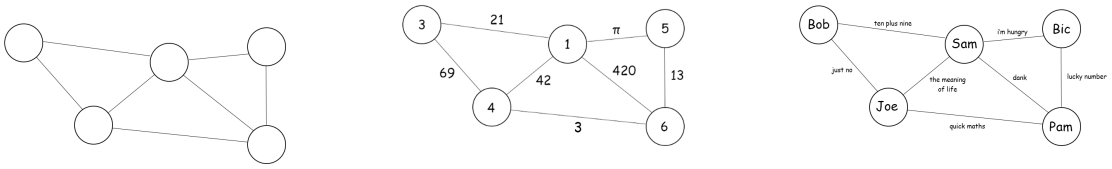
Aquesta disciplina ve molt marcada per la Teoria de Grafs, ja que en si mateix un sistema que utilitza aquesta tecnologia és un **graf** d'informació. Aquests estan presents en la vida quotidiana a tot arreu, tant les molècules químiques com un simple mapa del metro són representacions de la realitat utilitzant aquesta teoria.

2.3.1 Teoria de Grafs

En aquesta secció s'explicarà els conceptes bàsics de la teoria i com s'aplica al **deep learning**.

Un **graf** són dades estructurades en nodes i arestes. Els nodes contenen la informació i les arestes ens mostra com es relacionen els nodes entre ells mateixos. En la figura 7 es poden veure exemples molt senzills els quals ens permeten veure com és un graf,

com es representen els nodes i les arestes, com aquests poden prendre qualsevol classe d'informació i que tots dos elements tenen propietats.



(a) Graf en blanc

(b) Graf amb informació numèrica

(c) Graf amb informació alfabètica

Figura 7: Exemples bàsics de Grafs [Font: Towards Data Science]

També és important diferenciar entre els grafs direccionals i els que no ho són (Figura 8). En els direccionals cada aresta només connecta un node A cap a un B però no a la inversa (B cap a A) el que fa que es limiti (o es pugui controlar més) la circulació d'informació en el sistema, si és volgues fer arribar informació de B cap a A seria necessari afegir una altra aresta en el sentit contrari. En canvi en els que no són direccionals totes les arestes van en ambdues direccions permetent que la informació passi d'un extrem a un altre del graf sense cap impediment.

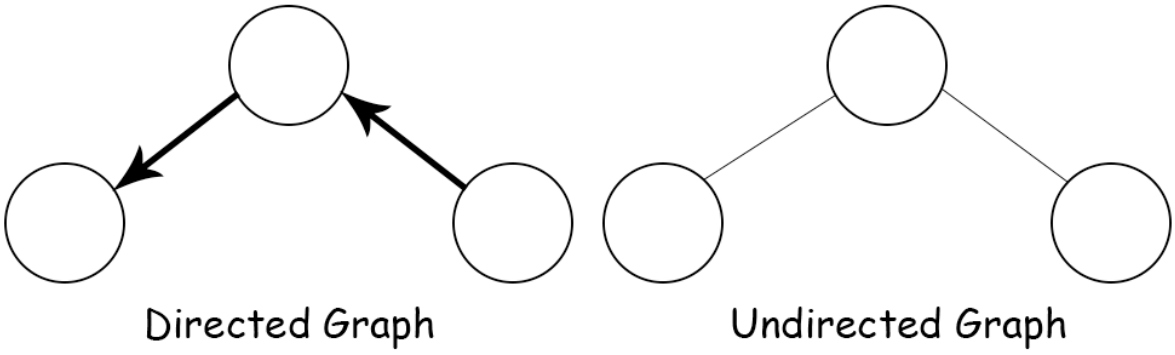


Figura 8: Diferència entre un Graf direccional i un no direccional [Font: Towards Data Science]

Altres maneres de classificar els grafs són:

- **Homogeni/Heterogeni:** Tots els nodes són iguals - Hi ha diferents tipus de nodes.

- **Estàtic/Dinàmic:** Tot es manté tal com comença - Hi ha modificacions de propietats tant de nodes com d'arestes i es poden afegir i treure qualsevol d'aquestes.

Un cop entès que és un graf és necessari saber com ho pot entendre un ordinador, ja que treballarem amb grafs implementats informàticament. Un ordinador treballa molt bé, amb les eines adequades, amb matrius que a les files tingui els nodes emissors i a les columnes els receptors essent els valors de l'interior de la matriu les característiques de les arestes. És necessari afegir que hi ha un altre tipus d'aresta que és la ponderada, que en aquest cas ens indica la fermesa amb què es relacionen un parell de nodes (en dona informació), i és la base del **deep learning**.

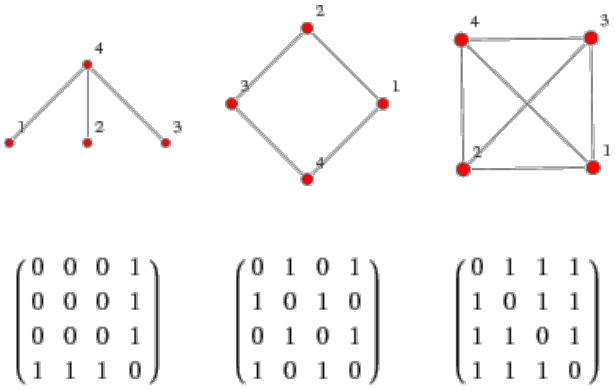


Figura 9: Representacions de grafs en matrius [Font: Towards Data Science]

En la figura 10 podem observar que els grafs no direccionals la matriu és completament simètrica, en els direccionals només tenim informació en la meitat superior de la diagonal i que els ponderats les matrius no són binaries sinó que els valors que es poden prendre són molt variats. També és interessant veure que mentre un node no es pugui connectar a ell mateix mitjançant una aresta la diagonal de la matriu sempre serà zero.

2.3.2 Aplicació al Deep Learning

Ja tenint clar que tot es basa en una xarxa neuronal, tornem a la base de com es construeix un sistema de **deep learning**. Tot comença amb l'equació d'una recta: $y = m \times x + b$ on **y** és la resposta, **x** és l'entrada, **m** és la ponderació (com a la Figura 10) i la **b** és el biaix (correcció d'activació). Amb això és possible determinar si la sortida està activada o no (si ha trobat el patró que busca).

Un cop s'ajunten tots els nodes i arestes el resultat és un esquema semblant al de la figura 12 on es pot veure la capa d'entrada de dades dues capes amagades i una de sortida. En el projecte actual la capa d'entrada ha de ser tan gran com **píxels** tingui la imatge captada per la càmera (en l'exemple seria una imatge de 3 **píxels**). Després la capa de sortida ha de tenir tants nodes com diferents objectes sigui necessari detectar (en

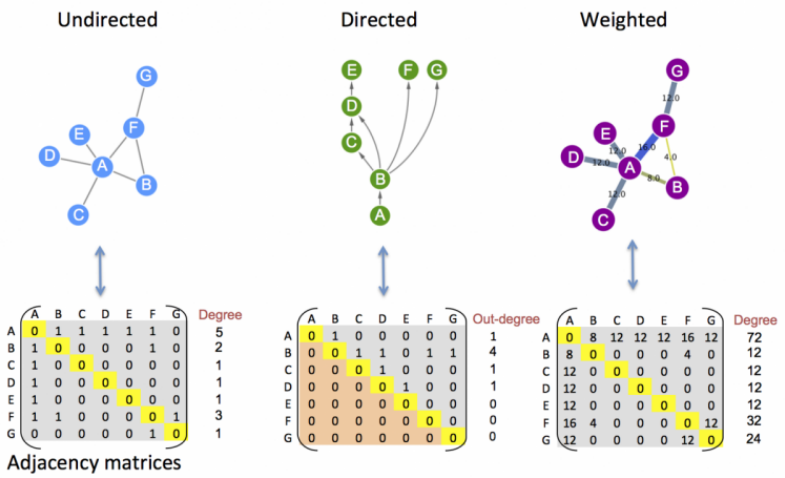


Figura 10: Tipus de representacions de grafs en matrius [Font: Towards Data Science]

l'exemple només pot detectar 1 cosa o no detectar res). I les capes amagades serveixen per a millorar l'eficàcia del programa.

Un cop està dissenyat el sistema, cal entrenar-lo per a tal que compleixi el seu objectiu de manera satisfactòria. Per això és necessari calcular la funció de pèrdua: $e_i = Y_i - \hat{Y}_i$ on Y_i és el resultat que se suposa que s'havia d'haver obtingut i $\hat{Y}_i$ és el resultat obtingut realment. L'objectiu és disminuir l'error tant com sigui possible, i s'aconsegueix variant en la mida necessària les ponderacions i biaix de cada node aresta del sistema. Hi ha una manera de fer exactament aquest procés, a través de diverses etapes d'entrenament, que es diu propagació inversa.

Per a fer la propagació inversa, cada valor s'ajusta corresponent després d'aplicar un diferencial vectorial (∇) a la funció de pèrdua i el resultat és com haurien de canviar els valors per a tal de minimitzar l'error (tal com s'explica en el vídeo de **3Blue1Brown**). I en realitat el que estem fent és fer derivades per a trobar un camí de baixadaçap a un punt més baix cada cop tal com es fa en una gràfica 2D però en aquest cas amb moltes més dimensions.

Com més quantitat de dades tinguis mentre s'entrena i més etapes, més precís serà l'ajust.

En resum:

- Els grafs es representen digitalment com matrius.
- El **deep learning** és una branca del **machine learning**.
- El sistema aprèn iterativament a mesura que se li dóna més informació.

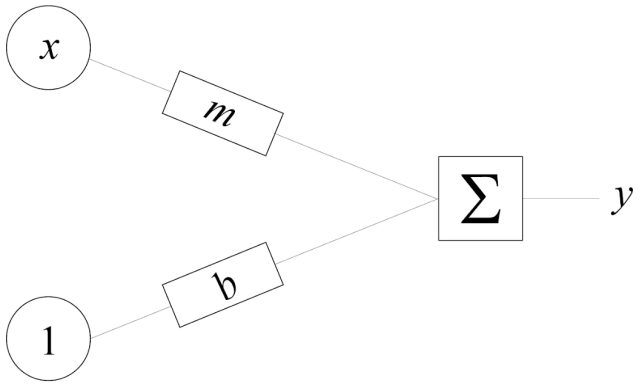


Figura 11: Cas general d'un node d'entrada i un de sortida amb biaix [Font: Towards Data Science]

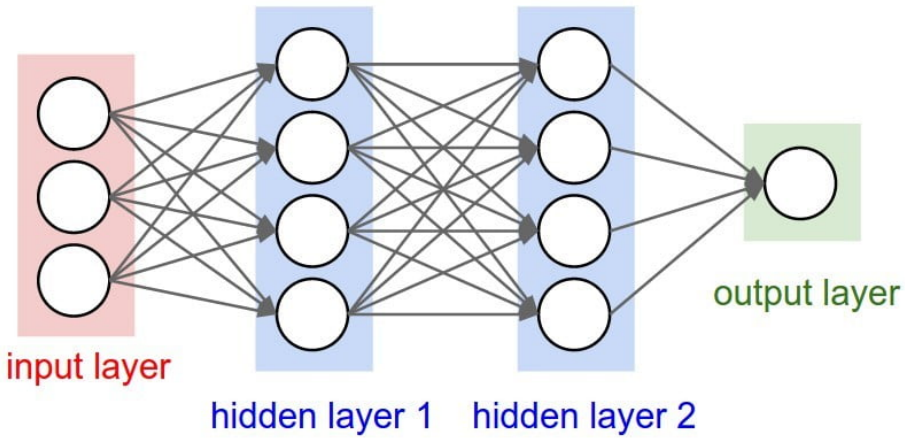


Figura 12: Exemple d'un sistema complet amb 3 entrades i 1 sortida [Font: Towards Data Science]

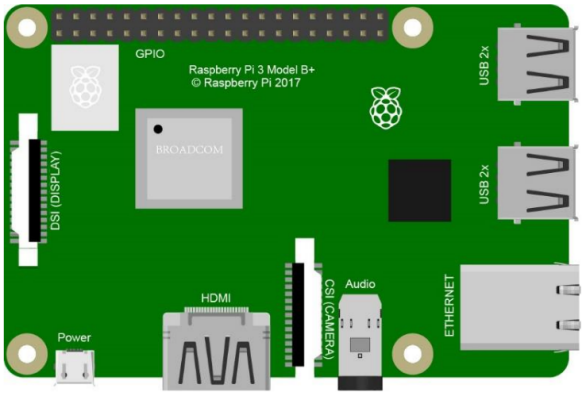
3 Maquinari

3.1 Raspberry Pi 3 B+

Per a poder realitzar qualsevol computació necessitem el **maquinari** que suporti tota l'operació.

La *Raspberry Pi 3 B+* (Figura 13) és una sola placa que conté un petit ordinador que pot funcionar independentment. Únicament són necessaris els **perifèrics** típics d'un ordinador de sobretaula: un ratolí, un teclat i una pantalla.

(a) Raspberry Pi 3 B+ [Font: Raspberry Pi]



(b) Diagrama de la Raspberry Pi 3 B+ [Font: Freenove]

Figura 13: Placa Raspberry Pi 3 B+

En cas que es vulgui tenir connexió a Internet també és necessari disposar d'una connexió via cable **Ethernet** o via **Wi-Fi**, ja que la placa té incorporada la capacitat de connexió sense fils. Gràcies a aquesta habilitat, també es pot operar a través d'un **escriptori virtual** en cas que no es disposi del material necessari i es tingui un ordinador connectat a la mateixa **LAN**.

3.1.1 Característiques

Normalment la placa està alimentada mitjançant un adaptador de corrent proporcionat per Raspberry Pi que permet l'alimentació de forma òptima. Però pel fet que és necessari instal·lar-la en un vehicle autònom i no és possible estar constantment connectat a la xarxa elèctrica, la Raspberry Pi 3 B+ rebrà l'energia a través de la *Smart Car Shield for RPi* (apartat 3.3), la qual està dissenyada per a aquest propòsit.

També és necessari recordar que la placa disposa de 40 pins **GPIO** que es poden fer servir per a comunicació amb l'exterior, cosa que el *Smart Car Shield for RPi* requereix

per a la connexió amb el port I^2C i també amb el senyal lluminós que disposa el vehicle.

GPU	Broadcom BCM2837B0
CPU	Cortex-A53 64-bit SoC @ 1.4GHz
Memòria	1GB LPDDR2 SDRAM
Connectivitat	Gigabit Ethernet — 5GHz 802.11 ac wireless LAN
Bluetooth	Bluetooth 4.2, BLE
USB	4 x USB 2.0 ports
Accessibilitat	Extended 40-pin GPIO header
Càmera	Port **CSI**
Emmagatzematge de dades	microSD
Mides	85mm x 56mm
Potència d'entrada	5V i 2.5A

Taula 1: Característiques de la Raspberry Pi 3 B+ [Font: Raspberry Pi]

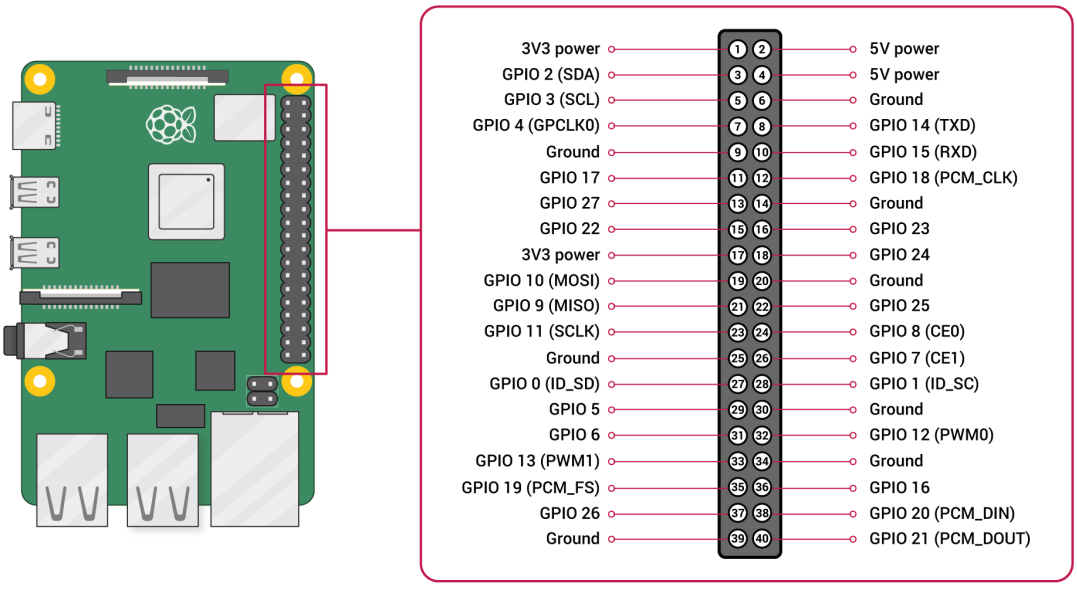


Figura 14: Esquema dels pins GPIO [Font: Raspberry Pi]

3.2 Vehicle

Per la naturalesa del projecte, el vehicle ha de disposar d'una càmera per disposar de l'**input** necessari per a percebre el seu entorn i reaccionar adequadament i, per últim, la

manera de processar tota la informació. També és recomanable disposar d'una regulació de velocitat per a aproximar millor a un model real de cotxe.

Per processar la informació ja he parlat a l'apartat **3.1** del fet que s'ha utilitzat aquesta placa perquè estava disponible per utilitzar. Aquest fet em va marcar a l'hora de buscar un vehicle que complís tots els requisits i alhora fos compatible amb una Raspberry Pi.

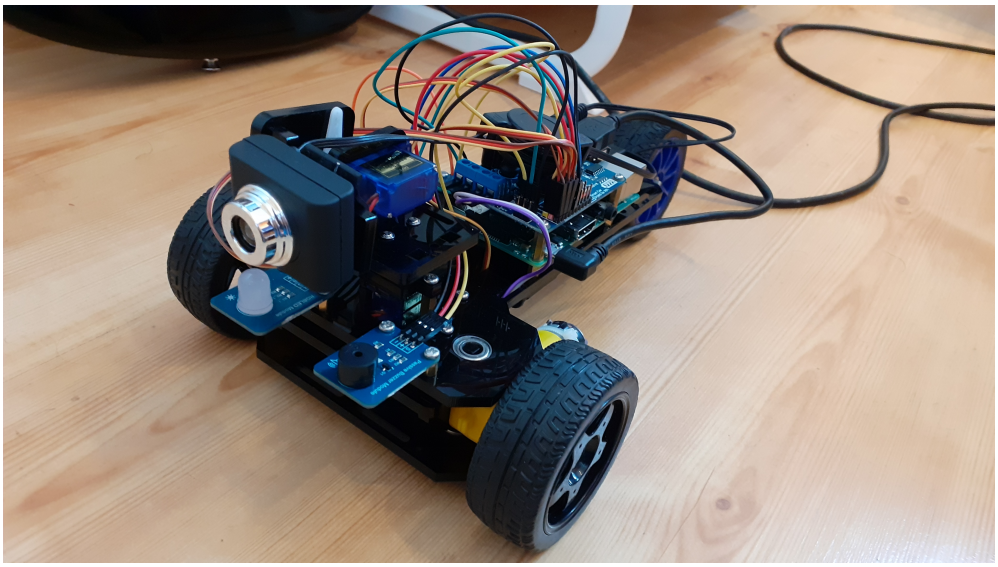


Figura 15: Three-Wheeled Smart Car [Font: Pròpia]

Finalment en vaig trobar un que complia perfectament amb les necessitats del projecte, ja que està dissenyat específicament per a treballar amb una Raspberry Pi (recomanant les 4B/3B+/3B), amb una càmera amb port **USB** i amb la capacitat de controlar dues de les tres rodes de què disposa de manera independent mitjançant el sistema **PWM**.

3.3 Smart Car Shield for RPi

Aquesta placa (Figura 16) té la capacitat treballar complementant la Raspberry Pi per a tal d'afegir a aquesta la capacitat de controlar totes les funcions de les quals disposa: controlar els motors propulsors, els **servos**, els senyals lluminosos i sònics i, si està instal·lat, el sensor ultrasònic.

3.3.1 Característiques

La placa està alimentada pel *DC power jack* a través de les dues bateries de 3,7V permetent donar energia a tot el sistema: motors, llums, càmera i la placa Raspberry Pi.

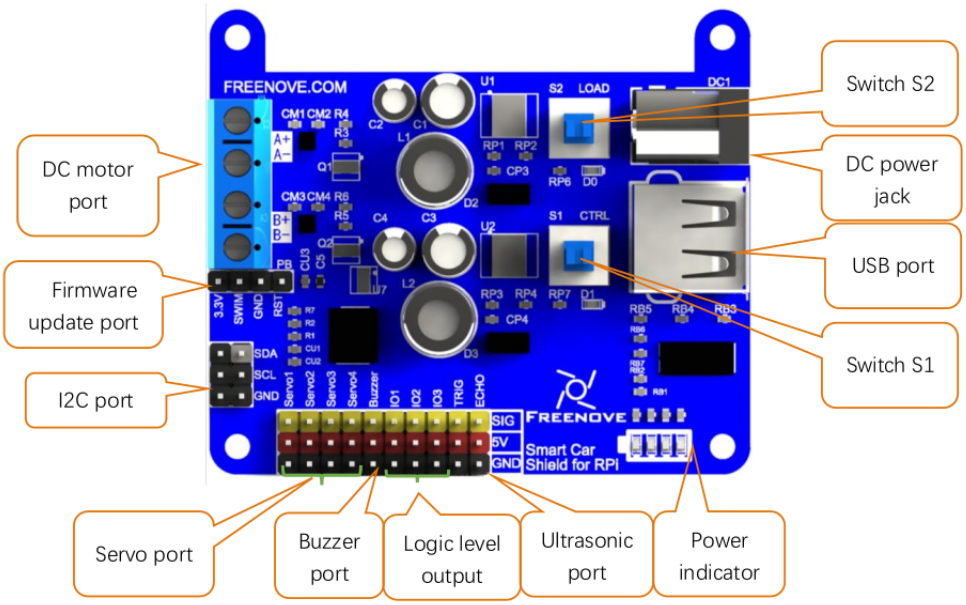


Figura 16: Placa controladora de les funcions del vehicle [Font: Freenove]

Aquesta última es comunica i alimenta amb la Shield a través del *USB port*.

També disposa d'interruptors (*Switch S1/2*) per a permetre el pas de corrent (*S1*) i un altre per a permetre el funcionament del vehicle (*S2*) però sense influir en la Raspberry Pi, connexions per a cada funció del sistema i un indicador de la bateria restant.

Per últim, té port I^2C per a poder comunicar-se amb protocol de comunicació I^2C, la informació en el qual es conserva encara que no hi hagi energia al sistema. Es connecta directament amb la Raspberry Pi 3 B+ mitjançant el **GPIO** d'aquesta última.

3.4 UNO R3

Ja que es vol implementar la detecció de semàfors al cotxe, és necessari fabricar un prototip a escala reduïda d'un semàfor i que actuï com a tal de manera que sigui possible testejar l'eficàcia de la detecció i la reacció del sistema en qualsevol de les situacions amb què es podria trobar un vehicle real.

Per aquest motiu és una necessitat disposar d'alguna manera de poder controlar l'encesa i apagada de cada llum d'un o més semàfors, a més d'altres llums que podrien representar l'aparador d'una botiga o altres tipus de distraccions, i s'ha triat per a aquesta missió la placa UNO R3 d'Elegoo per la gran versatilitat que disposa i amb suficients sortides per a ser possible treballar amb més d'un dispositiu a l'hora.

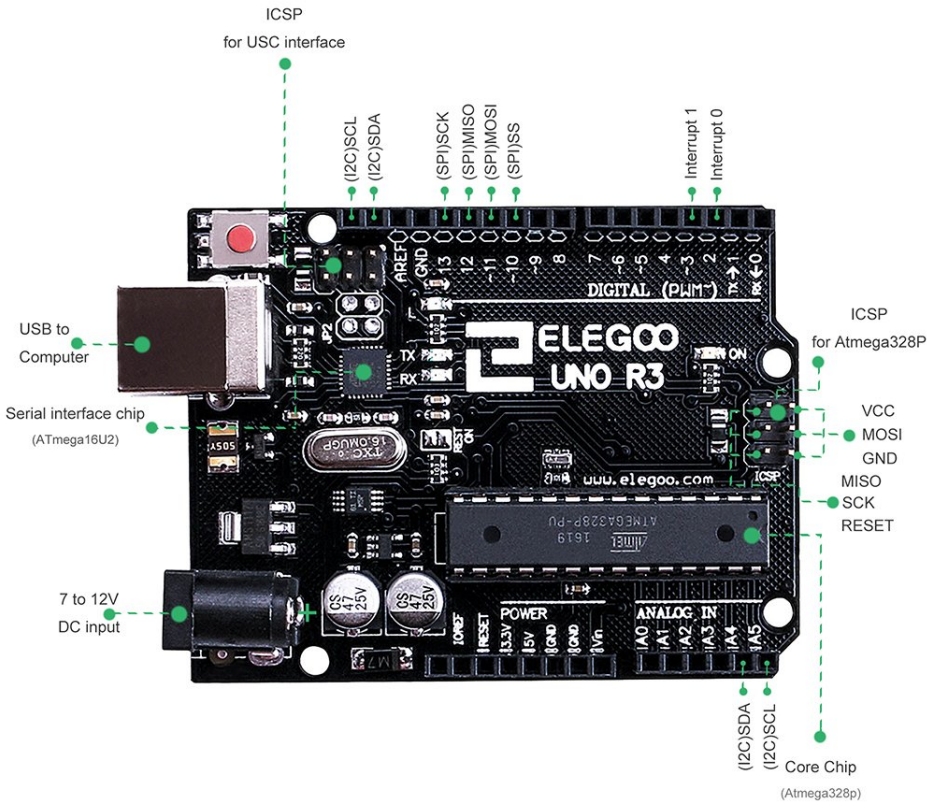


Figura 17: Placa Elegoo UNO R3 [Font: Amazon]

3.4.1 Característiques

És una placa compatible amb **Arduino IDE**.

Disposa de 14 pins digitals d'entrada i sortida i d'un *bootloader* (permetent carregar codi sense necessitat de cap **maquinari** programable extern).

Controlador	ATmega328P microcontroller
Voltatge d'entrada	7-12V
Accessibilitat	14 Digital I/O Pins (6 PWM outputs) ‖ 8 Analog Inputs
Connectivitat	Connexió **USB**

Taula 2: Característiques de la UNO R3 [Font: Amazon]

4 Programari

Tot **programari** està escrit en algun llenguatge de programació, ja sigui en **llenguatge màquina** o en alguns una mica més desenvolupats com podrien ser el **C++** o el **Python** els quals ja necessiten un **compilador** o intèrpret, respectivament, perquè la **CPU** pugui entendre les instruccions.

Per a aquest projecte s'utilitzarà el llenguatge **Python**, ja que és el llenguatge que utilitza el vehicle utilitzat i és compatible amb totes les eines utilitzades.

4.1 Eines

4.1.1 OpenCV

OpenCV (*Open Source Computer Vision Library*) és una de les biblioteques més utilitzades en COMPUTER VISION enfocada sobretot en aplicacions en temps reals.

Figura 18: Logo d'OpenCV [Font: Wikipedia]

Una de les característiques que ha fet que guanyés tanta popularitat des que es va publicar el 1999 és el simple fet de ser un codi lliure i, per tant, permetre adaptar el codi original a cada aplicació, objectius i programador. Aquest fet és possible gràcies al fet que està publicat sota la llicencia BSD que permet ser utilitzat lliurement tant personalment com comercialment i per a investigació.

Un altre dels grans punts forts d'aquesta biblioteca és la gran flexibilitat que presenta a l'hora de poder funcionar en diferents plataformes com poden ser *Linux, Os, Windows i Android*, diverses arquitectures com *x64, x32, ARM, mòbils i Raspberry Pi* i diferents llenguatges de programació com *C++*, ja que l'última versió està desenvolupada completament amb aquesta, però també està preparat per a ser utilitzat de forma senzilla i similar en *Python, Java, Matlab, Octave i Javascript*.

D'ençà que va ser publicat, la biblioteca ha guanyat molts usuaris que a l'hora han anat desenvolupant algoritmes i codi paral·lelament al desenvolupament oficial d'aquesta complementant-la. La gran quantitat d'usuaris al llarg del seu recorregut, també comporta que per a la gent que hi arriba nova a la comunitat, es trobi errors i problemes que gairebé segur algú altre ha trobat i solucionat.

Aquesta evolució i gran base d'usuaris ha portat al fet que grans empreses tecnològiques, com poden ser *Microsoft* o *Google*, o d'automoció, com *Honda* i *Toyota*, cosa que ajuda a augmentar la popularitat i la voluntat de millorar i l'ha portat a on es troba actualment.

OpenCV està dissenyada modularment per a poder utilitzar només les funcions necessàries. Aquesta característica no està aplicada a tots els llenguatges que suporta (com és el cas de Python), però si en C++.

A continuació hi ha alguns dels mòduls de què disposa:

- **core module:** Tal com indica el nom *core* (núcli), és la base de la biblioteca, els blocs sobre els quals es comença a construir qualsevol programa basat en OpenCV.
- **improg module:** *Image Processing* (Processament d'Imatge) serveix per a manipular les imatges i que s'adaptin a les necessitats com canviar de color, redimensionar i filtrar.
- **highgui module:** Serveix per a incorporar una **GUI** a les finestres com poden ser barres de selecció de valors.
- **imgcodecs module:** Per a llegir i guardar imatges.
- **videoio module:** Per a llegir i guardar vídeos.
- **calib3d module:** Ajuda a modelar un món 3D a partir de les imatges en 2D.
- **video module:** Algoritmes per a la detecció de moviment, seguiment o extracció de fons.
- **objdetect module:** Per a la detecció d'objectes.

4.1.2 TensorFlow

TensorFlow és de nou una eina de codi lliure desenvolupada per *Google* (concretament *Google Brain*) el 2017 orientat a **dataflow** molt utilitzat per a **machine learning**, a causa d'aquest fet, amb moltes aplicacions neuronals. En resum és una biblioteca enfocada a càlcul numèric d'alt rendiment.

Aquesta eina s'ha utilitzat per a una gran varietat d'aplicacions com en el reconeixement de veu i d'objectes, robòtica i qualsevol camp en què sigui necessari reconèixer

Figura 19: Logo de TensorFlow [Font: Planeta Chat Bot]

patrons.

Alguns exemples d'aplicacions són els següents:

- **RankBrain (Google):** És la xarxa neuronal utilitzada per a filtrar i ordenar bilions de pàgines en les cerques fetes a `www.google.com`.
- **Deep Speech (Mozilla):** És una implementació que rep el so de les paraules i les converteix a text. És capaç de reconèixer el Mandarí (que és una llengua tonal) com que el sistema aprèn a través d'espectrogrames els diferents tons que pot tenir la llengua. (Figura 20).
- **Fraud detection (PayPal):** PayPal utilitza TensorFlow per a reconèixer patrons de possibles fraus a l'hora de fer pagaments en línia i poder parar-ho abans que es pugui produir.
- **Airbnb:** Ho fa servir per a millorar l'experiència dels seus usuaris classificant les imatges i detectant objectes.
- **MRI (GE Healthcare):** Permet detectar possibles tumors cerebrals que hi pugui haver tan sols proporcionant-li la imatge d'una ressonància magnètica.

4.1.3 Arduino

Arduino és una combinació de **maquinari** i **programari** lliures que permeten crear circuits electrònics amb un microcontrolador (la placa Arduino). Que sigui lliure significa, en el cas del **maquinari**, que tant els diagrames com les especificacions són de domini públic i, per tant, poden ser replicats sense cap problema. En el cas del **programari**, és

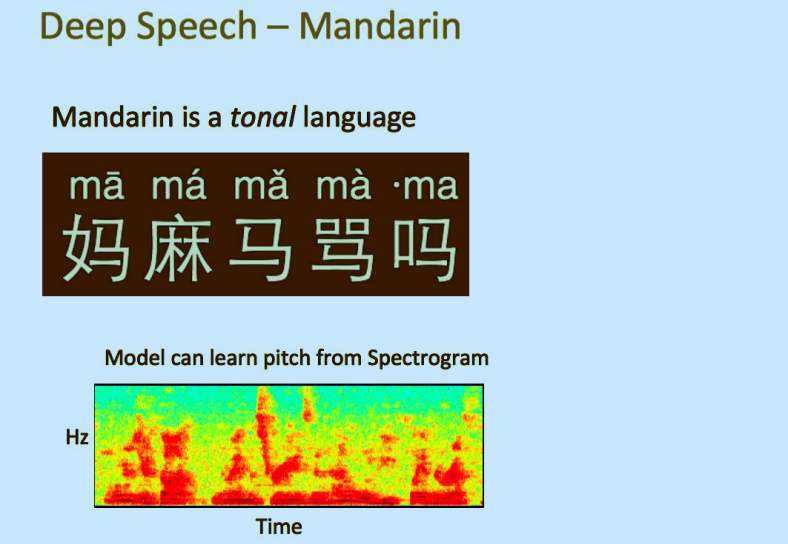


Figura 20: Representació de com pot aprendre una llengua TensorFlow [Font: DZone (Rinu Gour)]

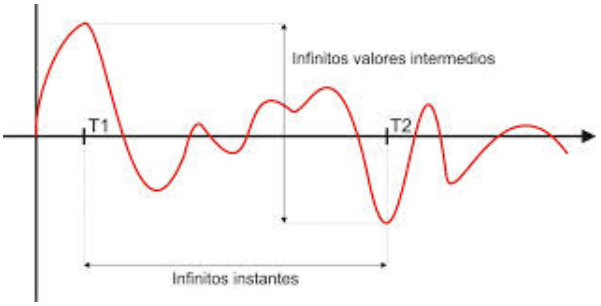
Figura 21: Logo d'Arduino [Font: Wikimedia]

tal com s'ha explicat a l'apartat 4.1.1.

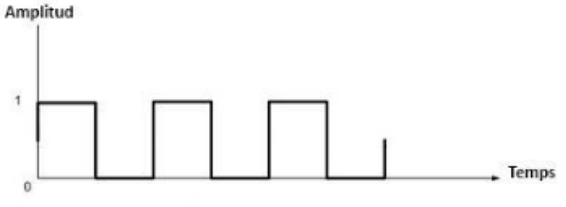
La placa disposa de ports d'entrada i sortida d'informació (*I/O ports*) que poden ser tant digitals com analògics i alguns de potència com poden ser sortides de diferents voltatges com entrada de terra. A la Figura 22 es pot apreciar com un senyal analògic pot prendre qualsevol valor mentre que el digital només uns predeterminats (en aquest tipus de sistemes acostuma a ser binari). D'aquesta manera cal utilitzar els pins necessaris en cada cas.

També és possible regular la sortida d'alguns pins digitals gràcies al sistema **PWM** permetent, per exemple, variar la intensitat de la llum que emet una llum connectada.

A continuació hi ha algunes aplicacions en què es pot trobar aquesta tecnologia:



(a) Gràfica d'un senyal analògica [Font: Viquipèdia]



(b) Gràfica d'un senyal digital [Font: Tecfluïd]

Figura 22: Comparació entre un senyal digital i una analògica

- **Bàscula:** Amb unes simples resistències piezoelèctriques i una pantalla **LCD** es pot construir una balança per a pesar per exemple fins a 200 kg sense molts problemes.
- **Jardinera:** Amb alguns sensors (com per exemple de llum, humitat i temperatura) pots controlar i mantenir constants aquests factors o anar variant-los segons les necessitats.
- **Detector d'empremtes digitals:** Amb una sola placa i un escàner d'empremtes és possible ajudar a protegir qualsevol cosa que un pugui pensar.

4.2 Implementació

Per al desenvolupament d'aquest projecte s'han utilitzat diferents mòduls i classes que permeten la detecció i posterior reacció del vehicle.

Primer cal detectar el semàfor. Per a assolir-ho s'han fet uns mòduls de detecció d'objectes, en primera instància amb TensorFlow i, posteriorment, amb OpenCV.

Un cop detectat on està ubicat el semàfor cal trobar l'estat que es troba aquest utilitzant els mòduls pertinents explicats a l'apartat 4.2.3 i variar el comportament del cotxe amb la classe *Car*.

Finalment s'ha escrit un petit codi referent al control de les llums del semàfor.

4.2.1 Detecció d'objectes

Per a poder detectar els semàfors s'utilitzarà l'**API** de TensorFlow *Object Detection*. És una eina desenvolupada en codi lliure, tal com és TensorFlow, i per tant és possible utilitzar-la i adaptar-la sense cap mena d'impediment a cada necessitat.

És un sistema desenvolupat per a facilitar la creació d'un detector d'objectes basat en TensorFlow. Tal com diuen a *Google*, aquesta aplicació la utilitzen a l'empresa per a detectar diferents objectes en una mateixa imatge (com mostra la Figura 23) i que els ha sigut molt útil per a les seves necessitats. També encoratgen a la gent a fer les seves contribucions.

Figura 23: Exemple de la Detecció d'Objectes [Font: Tensorflow Object Detection API]

Per a realitzar aquest model, s'ha partit des d'una estructura neuronal que ja havia estat entrenada anteriorment i es poden trobar a la pàgina del **zoo**. Aquestes han sigut entrenades a partir de diferents conjunts de dades com poden ser *COCO*, *Kitti*, *Open Images*, *AVA v2.1* i *iNaturalist Species Detection Dataset*.

Per exemple el conjunt *Open Images* disposa de milions de caixes de detecció i milions d'imatges etiquetades, entre moltíssimes altres característiques utilitzades per a aquestes finalitats, que permeten detectar la gran majoria d'objectes i entitats més comunes arreu del món a l'hora de permetre fer-ho de diferents maneres. Un clar exemple és la Figura 24.

En la Figura 24b es pot observar que no tan sols pot detectar una àguila, com també extreure la seva màscara (Figura 24a), comportant d'aquesta manera que és capaç de

(a) Màscara d'una àguila detectada.

(b) Detecció d'una àguila.

Figura 24: Diferents resultats d'aplicar un detector d'objectes [Font: Open Images]

detectar i extreure els límits de l'animal.

A partir de totes aquestes bases de dades, s'han creat un gran nombre d'estructures a partir de les quals és possible fer el teu propi programa de detecció i cada cop se'n creen de nous i es milloren els existents. A continuació (a la Taula 3) s'especifiquen alguns d'aquests models.

Model name	**Speed** (*ms*)	**COCO** *mAP*[1]	**Outputs**
faster_rcnn_nas	1833	43	Boxes
ssd_inception_v2_coco	42	24	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco	29	16	Boxes

Taula 3: COCO-trained models [Font: Tensorflow detection model zoo]

Els paràmetres especificats tenen el següent significat:

- **Speed:** El temps que tarda a processar la imatge i donar un resultat. Cal tenir en compte que aquests càlculs han sigut realitzats en imatges 600x600, una targeta gràfica *Nvidia GeForce GTX TITAN X* i en TensorFlow **GPU** v1.12.0.
- **COCO** *mAP*[1]**:** (*mean Average Precision*) La precisió què és capaç de detectar els objectes.

- **Outputs:** Quin tipus de detecció fa, màscara (24a) o rectangle al voltant d'aquest (24b).

Per a realitzar un detector d'objectes és necessari disposar d'imatges de l'objecte o objectes que es volen detectar. Hi ha dos camins que es poden seguir:

1. Obtenir les imatges de forma natural.

2. Crear un conjunt de dades a partir d'una sola imatge, o un nombre reduït d'aquestes, de l'objecte o objectes, i un altre conjunt d'imatges negatives (que no contenen l'objecte).

Tant si es tria un camí o un altre és necessari disposar d'un fitxer que indiqui en quina posició està cada objecte dintre de cada imatge quin dels tipus és (tant si s'entrena per detectar un sol objecte o més d'un).

Aquest és el punt on és necessari decidir quin camí seguir, ja que si es tria el primer, és més probable que sigui capaç de detectar més eficaç l'objecte perquè el sistema haurà sigut entrenat analitzant aquest en espais on posteriorment pot identificar-lo. Però també vol dir que cada imatge cal ser etiquetada, és a dir, especificar quins objectes hi ha, de quin tipus i en quina posició de la imatge. Hi ha una eina molt útil anomenada *LabelImage* desenvolupada per darrenl on et permet realitzar aquesta operació d'etiquetatge (Figura 25).

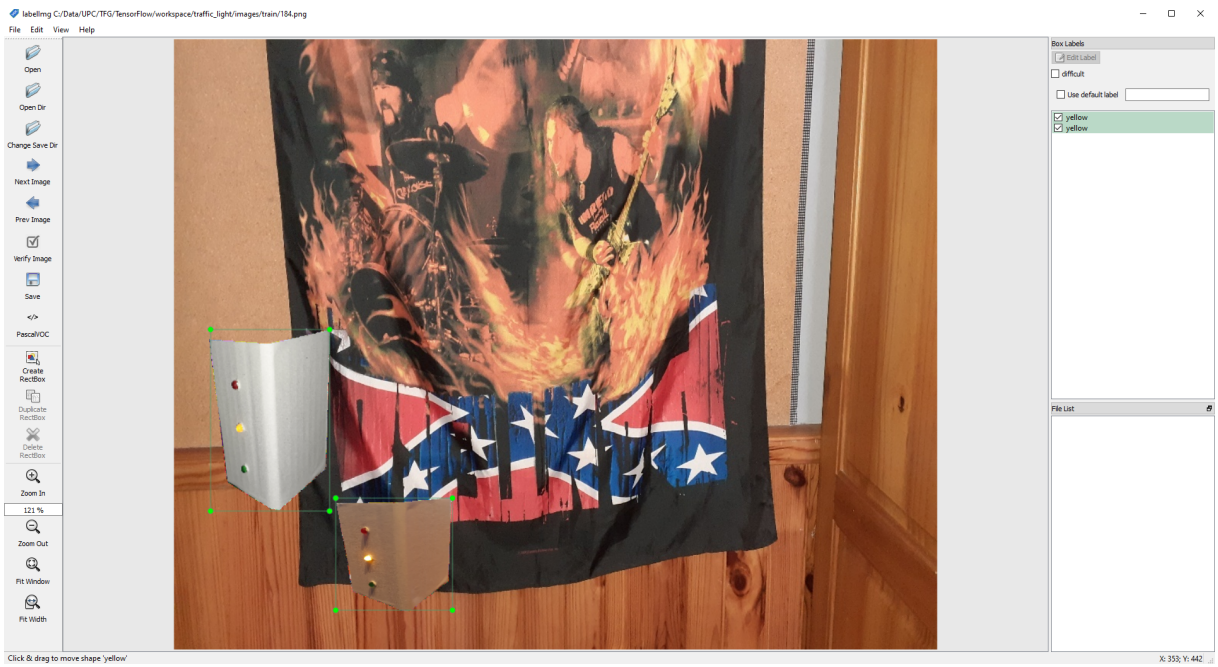


Figura 25: Captura de pantalla de *LabelImage* (darrenl a GitHub) [Font: Pròpia]

Un cop s'han etiquetat tots els objectes desitjats, aquesta informació s'emmagatzema en un fitxer *.xml* (un per cada imatge) i, posteriorment, es tradueix tota aquesta informació en un únic fitxer *.csv* que conté la informació de cada objecte de cada imatge tal com mostra el Codi 1. També s'acostuma a reservar unes quantes imatges (un 10% per exemple) per a fer els tests i comprovar l'evolució del procés d'entrenament.

```
filename,width,height,class,xmin,ymin,xmax,ymax
106.png,800,800,green,4,261,187,415
106.png,800,800,yellow,221,509,445,700
117.png,800,800,red,107,9,314,249
```

Codi 1:]Exemple de fitxer *.csv* [Font: Pròpia]

L'altre camí que es pot seguir, i és l'escollit per a aquest projecte, és el d'utilitzar imatges aïllades dels objectes i imatges utilitzades com a fons (*background*) i crear directament el fitxer *.csv*. D'aquesta manera, no cal realitzar tot el procés d'etiquetatge i, a vegades, equivocar-te en el tipus d'objecte o no marcar-ne algun que aparegui en la imatge.

Per a realitzar aquest pas s'ha escrit un fitxer basat parcialment en *partition_dataset.py* i *xml_to_csv.py* (disponibles a la pàgina del **Tutorial del Detector d'Objectes**) anomenat *generate_images_csv.py*. En aquest se li especifiquen els camins (*paths*) a seguir per arribar a on es troben els objectes (cada objecte separat en una carpeta diferent), les imatges de fons, on es volen guardar les imatges generades i el percentatge d'aquestes que seran de test i escriu el fitxer *csv* i guarda les imatges generades. És important esmentar que, tal com està escrit actualment, per a cada objecte ha d'haver-hi una màscara que permeti que només la part desitjada aparegui en la imatge.

Per ajudar a mantenir-lo més net i intel·ligible, part de les funcions utilitzades en l'últim programa es troben a part com poden ser la funció per a crear les noves imatges o evitar superposicions excessives.

La primera és la funció *combine* (Codi 2) que permet generar una imatge amb els objectes a sobre d'un fons, variant les seves mides i inclinant-los per a poder creant així les mostres destinades a entrenar el sistema.

```
bg, rectangles = combine(bg, list_tups, ratio=False)
```

Codi 2:]Funció *combine* [Font: Pròpia]

- **bg:** És la imatge de fons que se li proporciona i que es processarà fins a obtenir la mostra.
- **list_tups:** Una llista que conté totes les imatges amb les seves corresponents màscares per a ser implementades.

- **ratio:** Determina si cal mantenir la relació d'aspecte dels objectes.
- **rectangles:** Llista dels rectangles que contenen els objectes en la imatge final.

La segona és la presentada al Codi 3, on la funció d'aquesta és determinar si dues imatges se superposen més que un cert valor, és a dir, si els rectangles que defineixen aquestes coincideixen i creen un altre rectangle, es vol que aquest rectangle sigui rebutjat en cas que, el rectangle creat per la superposició, tingui una àrea inferior al 20% de la superfície de la imatge ja dibuixada.

Per saber si dos rectangles es troben hem de trobar si, en una coordenada, un d'aquests comença abans que acabi l'anterior. En la Figura 26 es pot veure més clarament com la coordenada x de Q1 es troba entre les coordenades x de P1 i P2 i el mateix succeeix amb la dimensió y.

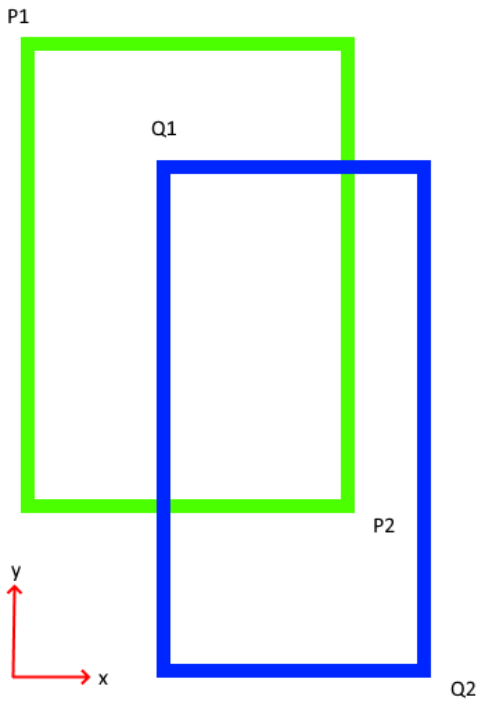


Figura 26: Diagrama de dos rectangles sobreposats [Font: Pròpia]

Un cop determinat si tots dos rectangles coincideixen, només cal comprovar quina és la superfície coincident. Primer cal definir l'àrea base que, en aquest cas, és la de la imatge que ja es troba dibuixada i que es trobarà en un índex inferior de la llista de rectangles (línia 61). Per a trobar l'àrea del rectangle de coincidència, cal utilitzar les coordenades que, ordenades tant ascendentment com descendentment, es trobin al mig tal com s'indica a la línia 69 del codi i com es pot observar en la Figura 26.

Un cop ja es té el rectangle desitjat es compara l'àrea d'aquest amb la base i en cas de ser superior al 20% es considera que no és acceptable. En cas positiu es tornarà a generar un rectangle nou i es tornarà a comprovar.

```
overlapping = overlap(l)
```

Codi 3:]Funció *overlap* [Font: Pròpia]

- **l:** Llista dels rectangles a mirar.
- **overlapping:** Determina si hi ha superposició. En cas afirmatiu retorna *True*.

A partir d'aquest punt, tant si s'ha triat un camí o un altre, és necessari transformar l'arxiu *.csv* a un *.record* perquè TensorFlow sigui capaç d'entrenar el sistema. Aquest pas es realitza mitjançant un codi escrit a la mateixa pàgina esmentada anteriorment anomenat *generate_tfrecord.py*, fent-lo servir tant per la informació de test com d'entrenament. Però per a poder executar-lo correctament, s'ha de definir el *label_map.pbtxt* on hi ha la informació de les etiquetes amb un número associat a cada una, que també cal adoptar a l'arxiu *generate_tfrecord.py*.

```
item {
  id: 1
  name: 'Red'
}

item {
  id: 2
  name: 'Yellow'
}

item {
  id: 3
  name: 'Green'
}
```

Codi 4:]Exemple de *label_map.pbtxt* amb tres etiquetes [Font: Pròpia]

Un cop finalitzada tota la preparació anterior només queda configurar un últim arxiu: el *pipeline.config*, on s'especifiquen els paràmetres que seguirà a l'hora d'entrenar el sistema. Aquest fitxer es troba dins de la carpeta descarregada en el zoo del Detector d'Objectes. Algunes de les línies a canviar es troben en el Codi 5, a la línia 3 cal definir el nombre d'objectes a detectar, a les línies 11, 13, 22 i 26 es defineixen les ubicacions els arxius creats amb anterioritat i per acabar, a la línia 7, es pot canviar el nombre d'imatges que s'agafen per lot, com més gran és aquest nombre, més memòria disponible serà necessari disposar.

```
model {
  ssd {
    num_classes: 90 # 3
.
.
train_config {
  batch_size: 24
.
.
train_input_reader {
  label_map_path: "PATH_TO_BE_CONFIGURED/mscoco_label_map.pbtxt" #
    annotations/label_map.pbtxt
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED/mscoco_train.record" # annotations/
    train.record
  }
}
eval_config {
  num_examples: 8000
  max_evals: 10
  use_moving_averages: false
}
eval_input_reader {
  label_map_path: "PATH_TO_BE_CONFIGURED/mscoco_label_map.pbtxt" #
    annotations/label_map.pbtxt
  shuffle: false
  num_readers: 1
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED/mscoco_val.record" # annotations/
    test.record
  }
}
```

Codi 5:]Part de l'arxiu *pipeline.config* [Font: Pròpia i TensorFlow Object Detection API]

La primera intenció ha sigut utilitzar el detector d'objectes per a detectar l'estat del semàfor a l'hora que l'objecte en si. Per a aquest propòsit s'ha utilitzat en primera instància el faster_rcnn_nas (de la Taula 3) a causa de l'alta precisió de la qual disposa, però, a l'hora de provar el resultat a l'ordinador es veu clar que no serà possible utilitzar-lo en el vehicle, amb molts menys recursos, ja que es veu entretallat. S'ha volgut provar d'utilitzar-lo en el cotxe, però consumeix més memòria de la disponible.

Per a reduir bastant els recursos emprats, es decideix utilitzar el següent model: el ssd_inception_v2_coco, el qual respon bé en l'ordinador. En aquest punt sorgeixen dos inconvenients: que detecta bé quan hi ha un semàfor però li costa diferenciar l'estat d'aquest i que, un cop instal·lat al cotxe, aquest respon molt lentament com indica la Figura 29.

Pel fet que aquest detector d'objectes no és molt precís diferenciant objectes sem-

blants, com poden ser els diferents estats d'un semàfor, s'ha optat per utilitzar el detector principalment per a ubicar el semàfor. I, després d'haver visualitzat com es veu a través del vehicle, s'ha decidit utilitzar imatges en escala de grisos, ja que sembla que es pot diferenciar millor el contorn i, en principi, ha d'ajudar a reduir el flux d'informació necessari a processar.

Finalment s'ha decidit utilitzar el ssd_mobilenet_v1_0.75_depth_quantized_coco perquè és més ràpid i té la menor precisió, però també vol dir que consumeix encara menys memòria per fer les deteccions. Aquest fet es pot comprovar com en la Figura 29 es veu una baixada molt gran entre aquest sistema i l'anterior. També s'ha comprovat que aquesta estructura no permet l'entrada d'imatges grises, així que és necessari mantenir els 3 canals d'aquestes i consumir molts més recursos dels necessaris.

Tot i haver decidit utilitzar l'estructura més convenient per a la capacitat de computació disponible, el resultat obtingut és encara massa lent per a les expectatives dipositades i es continuarà pel camí explicat a l'apartat 4.2.2.

Per a utilitzar el detector d'objectes s'ha agafat el codi proporcionat al web de **TensorFlow Object Detection API Tutorial** però utilitzant una funció diferent per a dibuixar els resultats en pantalla.

El Codi 6 mostra com hi ha dos paràmetres claus de la detecció com són *boxes* i *scores* (en cas d'haver-hi més d'un objecte a detectar també seria necessari utilitzar paràmetre *classes*) a la línia 4. La variable *boxes* conté les coordenades dels extrems dels rectangles que encapsulen els objectes detectats, però estan expressades en relació amb les dimensions de la imatge. Així que és necessari conèixer les dimensions amb què es treballa.

El cas de la variable *scores*, conté el percentatge de confiança que té el sistema d'haver detectat correctament i està ordenat de millor a pitjor, així que, com en aquest cas només interessa detectar un semàfor, s'agafa el primer de la llista. Per al paràmetre *classes*, hi conté la informació corresponent al tipus de detecció realitzada i és un nombre, el mateix especificat abans d'entrenar en el fitxer *label_map.pbtxt* (Codi 4) així que cal descodificar si s'utilitza.

```
1  def draw(img, param):
2  .
3  .
4      boxes, scores, *_ = param
5      score = scores[0][0]
6      if score > .5: # if the score is > 50%
7          ##[Get rectangle]
8          box = boxes[0][0]
9          ym, xm, yM, xM = box
10         xm = int(xm*w)
11         xM = int(xM*w)
```

```
12          ym = int(ym*h)
13          yM = int(yM*h)
14          ##![Get rectangle]
15  .
16  .
```

Codi 6:]Funció per dibuixar els paràmetres obtinguts de la detecció [Font: Pròpia]

En la Figura 27 es pot observar com detecta persones i cotxes en una fotografia que es podria trobar en qualsevol ciutat.

Figura 27: Exemple del resultat de passar una imatge pel detector d'objectes [Font: Vurbed New York]

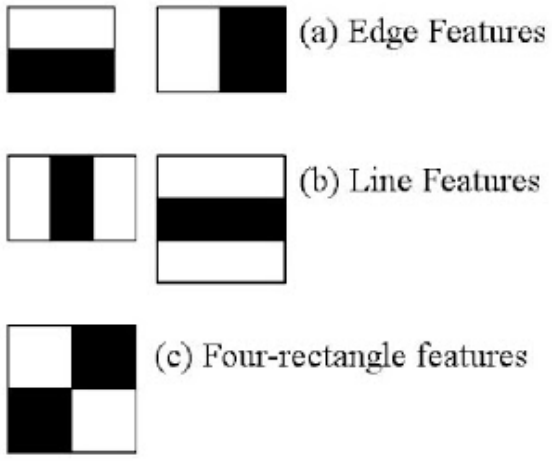
4.2.2 Detecció d'objectes (Cascade Classifier)

Com que el resultat amb la detecció d'objectes amb TensorFlow no ha sigut satisfactori, es torna a OpenCV amb el mòdul de detecció d'objectes (tal com s'ha comentat a l'apartat

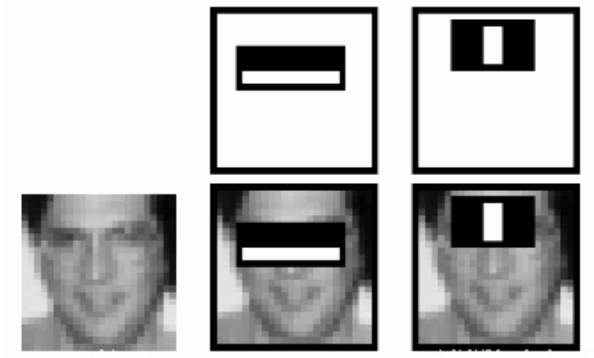
4.1.1).

Aquesta eina s'anomena Cascade Classifier utilitzant *Haar feature-based cascade classifiers* desenvolupat per Paul Viola i Michael Jones el 2001 amb el document *Rapid Object Detection using a Boosted Cascade of Simple Features* utilitzant imatges positives (de l'objecte) i negatives (qualsevol imatge que no contingui ni totalment ni parcialment l'objecte) mitjançant **machine learning**.

L'explicació del funcionament està basat en la detecció de cares frontals, així que serà necessari aconseguir moltes imatges positives (de cares) i de negatives (sense cares ni part d'elles) per a poder començar a entrenar el sistema. A partir d'aquestes, es comencen a extreure característiques (patrons) que permetran identificar cares en un futur. Les característiques *Haar* són les especificades en la Figura 28a que són equivalents als nuclis de **convolució**. Cada característica és un valor obtingut de restar la suma de píxels a la zona blanca a la suma de píxels a la zona negra ($negre - blanc$).



(a) Característiques



(b) Exemple de com troba els patrons

Figura 28: *Haar* [Font: OpenCV]

A partir d'aquí es calculen totes les possibles posicions i mides d'aquests nuclis i s'extreuen les característiques. A mesura que es van processant imatges i es tornen a fer passos, es van descartant molts patrons i consolidant d'altres. Per exemple a la Figura 28b es pot veure que el sistema detecta que els ulls acostumen a ser més foscos que la part de sota (nas i galtes) i que els ulls són més foscos que el pont del nas.

Un cop entrenat, només cal aplicar-ho i trobar les cares. El problema vindria quan s'haurien de comprovar milers i milions de característiques en cada posició de la imatge i amb diferents mides i seria una feina ineficient consumint un temps que no és viable tenint en compte que la gran majoria de vegades l'objecte buscat no ocupa tota la imatge, només

una petita part.

Per aquest motiu es va agafar la idea de fer-ho en cascada: fent etapes i, en cada una, només mirar certes característiques agrupades (normalment les primeres etapes contenen característiques molt petites) i que si una regió no és admesa en una etapa, aquesta ja no es mira en la següent. D'aquesta manera es descarta tota la regió que no hi ha cap possibilitat de contenir i s'inverteix el temps a estudiar on si pot haver-hi algun objecte. Aquest procés es pot veure en el vídeo *OpenCV Face Detection: Visualized* realitzat per A. Harvey (`https://vimeo.com/12774628`).

D'aquesta manera és possible detectar de forma ràpida una cara o qualsevol objecte per al qual s'entreni el sistema. També és molt menys precís que el sistema explicat a l'apartat 4.2.1, ja que té tendència donar molts més falsos positius. Per a prevenir o millorar aquest fet és necessari recol·lectar més imatges de l'objecte desitjat i utilitzar imatges negatives que tinguin a veure amb l'objecte: si es vol detectar un ocell volant, com més imatges de cel es tinguin més precís serà.

Per a construir-lo s'ha de seguir un procediment similar al del TensorFlow, ja que primer és necessari aconseguir les imatges per a, posteriorment, poder entrenar el sistema i es poden seguir els dos camins esmentats a l'apartat 4.2.1.

En comptes de tenir un fitxer *.xml* per a convertir-lo en un *.csv* i, després de l'entrenament, en *.record*, aquest mètode segueix un altre procediment. Primer se'n necessiten dos, un per a les imatges que contenen objectes i l'altre per a les negatives, del tipus *.lst* i *.txt*, respectivament.

La informació de les imatges negatives és el més senzill de generar, ja que només és necessari especificar el nom de la imatge (amb el camí relatiu fins a ella) tal com indica el Codi 7. Es pot observar com les imatges estan emmagatzemades sota la carpeta *neg*.

```
1 neg/1.jpg
2 neg/10.jpg
3 neg/100.jpg
4 neg/1000.jpg
5 neg/1001.jpg
6 neg/1002.jpg
```

Codi 7:]Exemple del fitxer d'imatges negatives [Font: Pròpia]

Per a la informació positiva, hi ha més paràmetres a tenir en compte. Primer cal especificar el nom de la imatge i, si escau, el camí relatiu, seguidament el nombre d'objectes presents en aquesta i, per finalitzar, tants rectangles com objectes. Com es pot observar al Codi 8, un rectangle està compost per quatre números que corresponen, respectivament, al punt superior esquerra $(x, y) = (0, 7)$ i a les dimensions del rectangle en amplada i alçada $(w, h) = (39, 52)$.

```
1.png 1 0 7 39 52
2.png 1 26 17 38 51
3.png 1 9 14 38 52
4.png 1 32 26 30 41
5.png 1 35 34 34 46
6.png 1 23 39 40 54
```

Codi 8:]Exemple del fitxer d'imatges positives [Font: Pròpia]

Per a obtenir el fitxer de les imatges positives, es pot fer de diferents maneres:

1. A mà amb imatges naturals com s'ha esmentat en el cas del TensorFlow.
2. Utilitzant la funció d'OpenCV *opencv_createsamples* (opció no disponible en el moment de realitzar el projecte) per a generar imatges positives a partir d'una positiva i diverses negatives.
3. Creant un codi que permeti un resultat similar al del punt 2.

Per a generar aquestes imatges i fitxers, s'ha agafat part de codi d'en sentdex de la pàgina **Creating your own Haar Cascade OpenCV Python Tutorial** d'on s'han descarregat milers d'imatges negatives i creat el *neg.txt*.

Per a crear la informació de les positives, s'ha desenvolupat la funció *create_pos* del fitxer *download-image-by-link.py*. Aquesta funció està basada en la que s'utilitza per a generar les imatges per a TensorFlow però canvia el format a l'hora d'escriure la informació i que, en aquest cas, només hi ha un objecte per imatge.

Un cop es tenen les imatges i la informació d'aquestes, és moment de convertir-la en un fitxer tipus *.vec* que serà l'utilitzat per l'ordinador per a entrenar el detector. En condicions normals aquest pas es realitza utilitzant la mateixa funció *opencv_createsamples* esmentada abans, però aquesta no ha estat disponible i s'ha buscat una alternativa. L'eina trobada ha sigut el **Haar Cascade Training on Windows by GUI Tool** que ha permès obtenir els fitxers necessaris.

Per finalitzar només queda realitzar l'entrenament. Aquest procés proporciona un arxiu *.xml* que és el que utilitza OpenCV per a realitzar les deteccions. Per a aconseguir-lo s'ha utilitzat el mateix programa d'abans però també s'hauria de poder aconseguir mitjançant *opencv_traincascade*.

Com es pot observar en la Figura 29, s'han anat millorant cada cop més els temps de resposta del vehicle fins a aconseguir que cada cicle de detecció duri menys de mig segon, però hi ha un cert temps que tarda el sistema a preparar i carregar totes les variables necessàries que provoca que les imatges tinguin lloc, dins del cotxe, al cap d'un segon o un temps semblant.

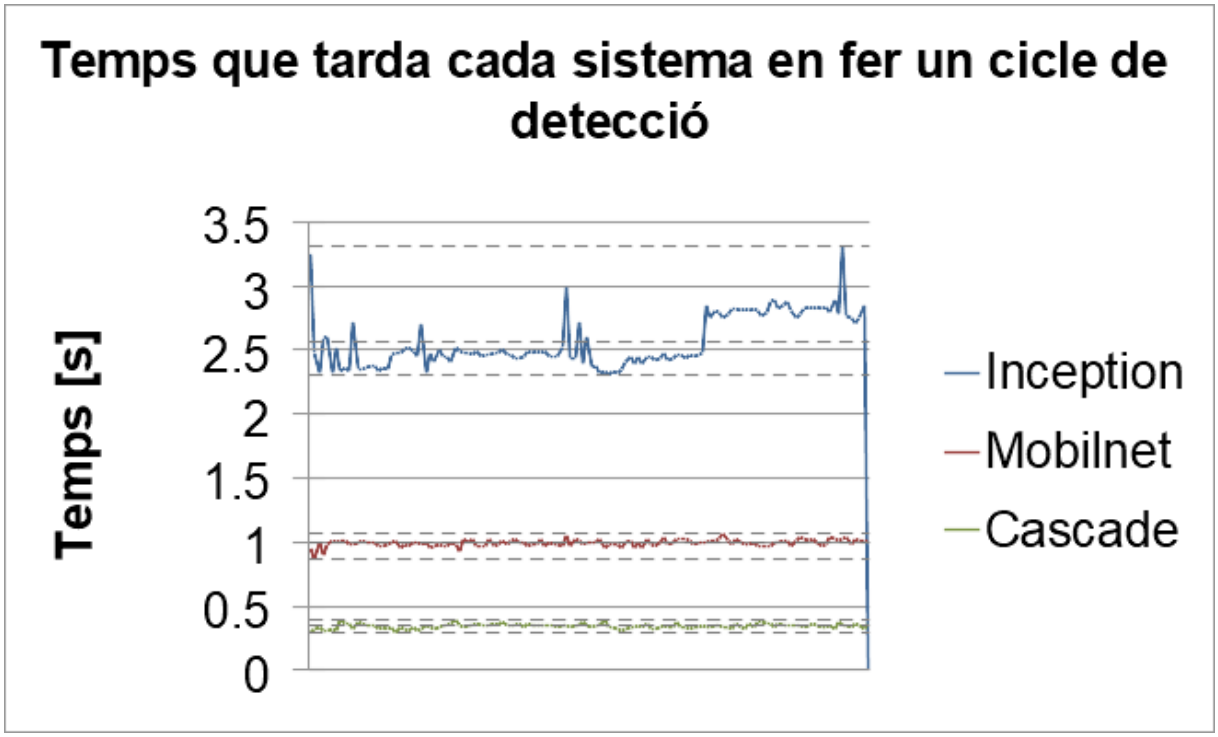


Figura 29: Gràfic dels diferents temps que tarda cada estructura en realitzar un cicle de detecció [Font: Pròpia]

Aquest petit retard, que en escales reduïdes és enorme, pot solucionar-se amb un equip una mica més potent (que disposi de més d'1GB de memòria RAM).

També és notable la diferencia de precisió que hi ha entre un sistema i un altre, amb l'actual de tant en tant detecta un fals positiu com es mostra en la Figura 30. Acostuma a passar quan el sistema detecta llum concentrada.

4.2.3 Detecció de l'estat del semàfor

Un cop detectat el semàfor i es té el rectangle que el conté, ja només queda determinar l'estat.

En primera instància s'ha pensat a detectar els LEDs dins dels límits detectats com a semàfor i determinar quin d'aquests té una intensitat lluminosa major i, com tots els semàfors tenen la mateixa disposició, determinar l'estat d'aquest. Però per culpa de la qualitat d'imatge de la càmera és inviable intentar trobar-los.

Per a aconseguir això s'han separat les imatges per gammes de colors. Primer de tot, en tenir canals de verd i vermell (en RGB), s'han extret aquests. Seguidament s'ha aïllat el color groc amb la funció *inRange* entre un rang de colors entesos com a grocs.

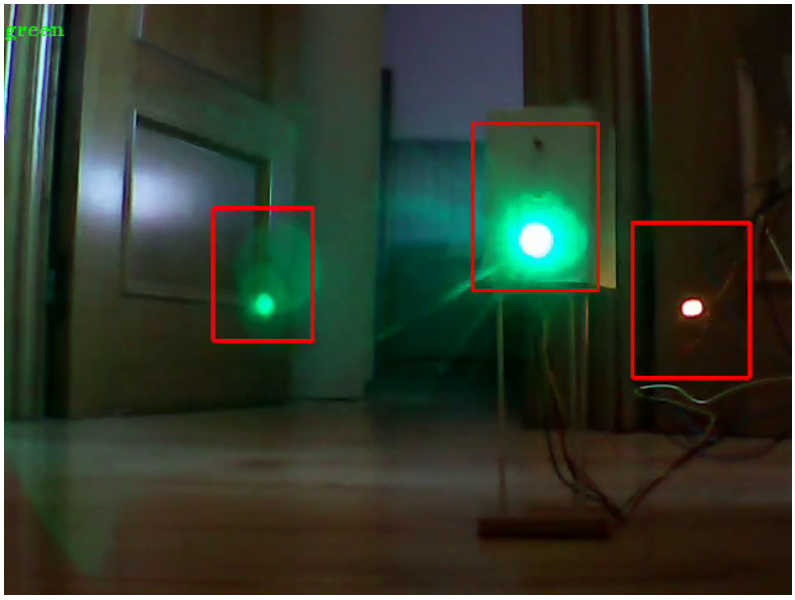


Figura 30: Detecció de falsos positius [Font: Pròpia]

I, per acabar, es calcula el valor de cada color per separat utilitzant per a aquest procés l'equivalència del valor que tindria a un píxel i, dels tres obtinguts, el que tingui un valor més elevat és el que es determina com a dominant.

Aquest color es troba mitjançant tres funcions del fitxer *modules.py*. La primera s'anomena *crop* (Codi 9) i determina la potència de cada color. Primer de tot s'aïllen els canals ver i vermell i es determina el valor mitjà d'aquests mitjançant la funció *pixel* (que redueix la imatge a un píxel i retorna el valor d'aquest). Després s'utilitza la funció *inRange* per a aïllar l'últim color (el groc) i s'hi aplica el mateix procediment que amb els altres dos colors.

```
1 red, green, yellow = crop(img, rect)
```

Codi 9:]Funció per determinar la potencia de cada color [Font: Pròpia]

- **img:** La imatge original (en color) amb la que es començarà el procés.
- **rect:** El rectangle que conté el semàfor.
- **red, green, yellow:** El valor de cada una de les gammes.

La segona funció és l'especificada en el Codi 10 que avalua cada franja de color i retorna una llista amb els colors amb valor màxim (en cas d'haver-hi més d'un color amb el mateix valor màxim).

```
color = color_matches(colors)
```

Codi 10:]Funció per comptar els cops que un color ha sigut dominant [Font: Pròpia]

- **colors:** Iterable dels valors de cada gamma obtinguda amb la funció anterior (amb el mateix ordre).
- **color:** Llista dels colors candidats trobats en aquella franja de la imatge.

I, per últim, està la funció *color_decider* (Codi 11) que determina un únic estat del semàfor tenint en compte les següents prioritats: el vermell té la màxima prioritat, seguit pel groc i, per últim, el verd. En cas de no haver detectat cap semàfor, no hi haurà valor i, per aquest motiu, a la línia 3 s'especifica que, si el màxim és zero, vol dir que no ha detectat res.

```
color = color_decider(colors)
```

Codi 11:]Funció per decidir el color dominant [Font: Pròpia]

- **colors:** Diccionari amb els colors com a claus i el nombre de vegades detectat cada un com a valor.
- **color:** El color final resultant.

4.2.4 Control del vehicle

Per a poder controlar el vehicle s'ha escrit un codi a partir de les funcions que l'empresa fabricant del model de cotxe proporciona (*mDEV*).

S'ha creat una classe anomenada *Car* heredada de la classe *mDEV*. Amb la nova és possible controlar els moviments de càmera, la direcció, la potència transmesa a les rodes i el color del LED. També seria senzill incorporar l'ús del brunzidor, però, com que no s'utilitza en aquest projecte, no s'ha implementat.

També hi ha limitats els angles que es poden disposar tant els servomotors de la càmera com els que controlen la direcció del vehicle. Aquests valors són fàcilment modificables per a adaptar-se a cada nou projecte que es pugui presentar. També seria una possibilitat tenir emmagatzemats aquests valors en un arxiu *.txt* i agafar els valors d'aquest.

Les funcions *go_fast* i *go_slow* són les més utilitzades, ja que permeten fer moure el vehicle. En el cas de la primera és l'equivalent a tenir via lliure sense cap impediment i va de pressa. Per a la segona, si és necessari anar més lent, el vehicle redueix la velocitat però, en el cas d'anar més lent no s'accelerarà com a mida de prevenció.

4.2.5 Comparació dels detectors d'objectes

Un cop implementats els dos models de detecció d'objectes, s'han buscat les diferències estadístiques entre els dos models i, cadascun, en diferents condicions.

S'han buscat diferents parelles de condicions a tenir en compte i, a l'hora, en cada una d'aquestes s'ha comprovat la precisió del detector d'objectes i de la conclusió a la qual arriba el sistema de l'estat d'aquest.

Els factors canviants han sigut:

- El detector utilitzat.
- Si el cotxe es troba en una posició allunyada o més propera.
- Si hi ha llums de distracció.

Cal comentar que, en el cas del resultat obtingut sobre l'estat del semàfor, el color groc es considerarà com a bo en cas que el detecti com a vermell, com que, en el cas del detector *Cascade*, aquest té tendència a detectar més d'un objecte per cicle de detecció i que es contaran com a deteccions correctes si els rectangles contenen bona part del semàfor.

A l'annex hi ha l'arxiu Excel on hi ha totes les dades. Però cal destacar que el detector basat en TensorFlow és més precís tal com es pot comprovar a la Taula 4. També es pot comprovar com no detecta del tot malament l'estat del semàfor, però si la posició d'aquest.

TensorFlow		**Cascade**	
95.80%		76.68%	
Objecte	**Estat**	**Objecte**	**Estat**
95.81%	95.80%	66.66%	86.71%

Taula 4: Taula comparativa entre els detectors d'objectes

A la Taula 5 es pot observar com la precisió del TensorFlow baixa molt, però que el percentatge és el mateix tant en la detecció com en l'estat. Aquest fet és degut a que hi ha alguna imatge que no el detecta, però que ho ja ho fa a la següent. En els casos de detecció hi ha un 100% d'encert.

Objecte	**Estat**
87.23%	87.23%

Taula 5: Percentatges d'encert del TensorFlow quan es troba allunyat sense distraccions.

4.2.6 Control de llums

Per a controlar els llums dels semàfors, s'utilitzarà el llenguatge Arduino amb la seva placa corresponent on, prèviament, s'haurà gravat el programa. Aquest està basat en el llenguatge **c++**.

El llenguatge Arduino està preparat per a ser utilitzat en sistemes d'execució ïnfinita". És a dir que no té un temps d'execució preestablert i aquest continuarà fins que deixi d'arribar energia al controlador.

Aquest es basa en dues funcions principals i bàsiques: el *setup* i el *loop*. La primera és la funció executada cada cop que s'encén el sistema i la segona és la que s'executa cíclicament un cop encès. El programa resultant en **C++** que permet controlar un semàfor es mostra a continuació.

```
1  //** Deffinitions **//
2  int red = 2;    //Define the red pin
3  int yellow = 4; //Define the yellow pin
4  int green = 6;  //Define the green pin
5
6  //** Program **//
7  void setup() {
8    // put your setup code here, to run once:
9    pinMode(green, OUTPUT); //Declare gren pin as an output
10   analogWrite(green, 157);  //Set green to half intensity
11
12   pinMode(yellow, OUTPUT);  //Declare yellow pin as an output
13   digitalWrite(yellow, LOW); //Turn on yellow
14
15   pinMode(red, OUTPUT);
16   digitalWrite(red, LOW);
17 }
18
19 void loop() {
20   // put your main code here, to run repeatedly:
21   analogWrite(green,157); //turn on green LED
22   delay(5000);            //wait 5 seconds
23   digitalWrite(green,LOW);  //turn off green LED
24
25   digitalWrite(yellow,HIGH);  //turn on yellow LED
26   delay(2000);                //wait 2 seconds
27   digitalWrite(yellow,LOW); //turn off yellow LED
28
29   digitalWrite(red,HIGH); //turn on red LED
30   delay(5000);            //wait 5 seconds
31   digitalWrite(red,LOW);  //turn off red LED
32 }
```

Codi 12:]Codi per a controlar les llums del semàfor [Font: Pròpia]

5 Conclusions

Un cop realitzat aquest projecte i haver après moltes nocions noves sobre el *Machine Learning*, *Deep Learning*, *Computer Vision* i la conducció automàtica, cada cop és més evident que aquest és el futur de l'automoció, ja que permetria evitar molts accidents provocats per errors humans.

Després d'haver dissenyat i construït el model, s'han arribat als objectius marcats al començament del projecte amb alguns matisos que es poden, i s'han de continuar, millorant.

L'objectiu era permetre que un vehicle detectés, mitjançant una càmera, un semàfor i actués d'acord amb les normes de circulació i s'ha complert.

Encara que s'ha de millorar la robustesa del sistema de detecció, sempre s'ha de millorar, ja que el detector d'OpenCV és més propens a donar alguns falsos positius. I si s'utilitza el de TensorFlow, hi ha un temps de resposta superior. És convenient trobar un punt entremig.

6 Proposta de futur

Hi ha molt treball per fer per a aconseguir un vehicle totalment automàtic. Hi ha bastants camins clars per a poder continuar millorant el model.

- Disposar d'un processador més potent capaç de processar la informació en un temps menor. També seria convenient provar-ho a escala i distancies reals per a tal de veure si realment és viable utilitzar-ho. Tenir més de 1GB de RAM que hi ha actualment.
- Per a poder utilitzar-ho a una escala més gran seria necessari disposar d'una càmera amb més resolució per a poder visualitzar i detectar correctament i amb prou definició perquè amb l'actual si capta el semàfor amb prou resolució ja està massa a prop.
- Poder tenir més càmeres per a poder observar millor l'entorn i, possiblement, disposar d'un detector sònic per a evitar col·lisions.
- Disposar moltes més imatges en diferents condicions dels semàfors i entrenar el sistema tantes hores com sigui necessari, tant si s'utilitza TensorFlow com OpenCV.
- Implementar el sistema de seguiment de línies per a mantenir el vehicle en un carril i, per a aquesta aplicació, es podria utilitzar de manera bastant eficient el sistema dissenyat pel senyor Cristian Alonso Vallejo en el seu estudi **Design and Implementation of an Autonomous Driving System for a RC Car**.

7 Impacte mediambiental

El món de l'automoció està evolucionant cada cop més cap als cotxes completament elèctrics per culpa de la contaminació provocada pels derivats del petroli i l'escassetat d'aquest. També és comú que els vehicles que s'estan desenvolupant siguin elèctrics, cosa que s'adapta perfectament al futur del sector.

L'impacte mediambiental directe de l'energia elèctrica és gairebé nul. Però sí que té un impacte és la fabricació dels components de les bateries que emmagatzemen l'energia i, sobretot, del procés que reben els components quan deixen de ser útils. Encara que segueix sent menys contaminant que el model actual de desplaçament.

Bibliografia

3Blue1Brown, *What is backpropagation really doing? — Deep learning, chapter 3*. Youtube [En línia] Disponible: https://www.youtube.com/watch?v=Ilg3gGewQ5U&feature=youtu.be. 2017 [Accedit el 10/4/2020].

A. Gonzalez, *¿Qué es Machine Learning?*. cleverdata [En línia] Disponible: https://cleverdata.io/que-es-machine-learning-big-data/. [Accedit el 9/4/2020].

A. Harvey, *OpenCV Face Detection: Visualized*. Vimeo [En línia] Disponible: https://vimeo.com/12774628. 2010 [Accedit el 17/6/2020].

A. Plitt, *New York City's streets are 'more congested than ever': report*. Curbed New York [En línia] Disponible: https://ny.curbed.com/2019/8/15/20807470/nyc-streets-dot-mobility-report-congestion. 2019 [Accedit el 17/6/2020].

A. Rosebrock, *Rotate images (correctly) with OpenCV and Python*. pyimagesearch [En línia] Disponible: https://www.pyimagesearch.com/2017/01/02/rotate-images-correctly-with-opencv-and-python/. 2017 [Accedit el 21/5/2020].

Amazon, *Elegoo ATmega328P ATMEGA16U2 UNO R3 - Placa con cable USB para Arduino*. [En línia] Disponible: https://www.amazon.es/Elegoo-ATmega328P-ATMEGA16U2-UNO-R3/dp/B01EWOE0UU. [Accedit el 9/4/2020].

Amazon, *Freenove Three-Wheeled Smart Car Kit for Raspberry Pi 4 B 3 B+ B A+, Robot Project, Tutorial and Code, Android App, Video Camera Ultrasonic Servo Wi-Fi Wireless RC*. [En línia] Disponible: https://www.amazon.es/Freenove-Three-wheeled-Raspberry-Detailed-Ultrasonic/dp/B06W54XC9V/ref=sr_1_9?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&dchild=1&keywords=freenove&qid=1592037443&sr=8-9. [Accedit el 18/3/2020]

BBVA, *Machine learning: What is it and how does it work?*. [En línia] Disponible: https://www.bbva.com/en/machine-learning-what-is-it-and-how-does-it-work/. 2019 [Accedit el 9/4/2020].

C. Alonso Vallejo, *Design and Implementation of an Autonomous Driving System for a RC Car*. [En línia] Disponible: https://upcommons.upc.edu/handle/2117/128039. Barcelona: Escola Tècnica Superior d'Enginyeria Industrial de Barcelona, UPC, 2018.

Curiousily, *Object Detection on Custom Dataset with TensorFlow 2 and Keras using Python*. [En línia] Disponible: https://www.curiousily.com/posts/object-detection-on-custom-dataset-with-tensorflow-2-and-keras-using-py

thon/. 2019 [Accedit el 29/5/2020].

dreamdragon, giuliano0, marksandler2, nealwu, pkulzc, sguada, srjoglekar246, tombstone, *Tensorflow detection model zoo*. GitHub [En línia] Disponible: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md. [Accedit el 2/6/2020].

darrenl, *LabelImg is a graphical image annotation tool and label object bounding boxes in images*. GitHub [En línia] Disponible: https://github.com/tzutalin/labelImg. [Accedit el 6/5/2020].

Evan, *Tutorial to set up TensorFlow Object Detection API on the Raspberry Pi*. GitHub [En línia] Disponible: https://github.com/EdjeElectronics/TensorFlow-Object-Detection-on-the-Raspberry-Pi. [Accedit el 7/6/2020].

F. Tong, *Everything you need to know about Graph Theory for Deep Learning*. Towards Data Science [En línia] Disponible: https://towardsdatascience.com/graph-theory-and-deep-learning-know-hows-6556b0e9891b. 2019 [Accedit el 10/4/2020]

Freenove, *Tutorial.pdf*. [En línia] Disponible: https://github.com/Freenove/Freenove_Three-wheeled_Smart_Car_Kit_for_Raspberry_Pi/archive/master.zip. [Accedit el 20/3/2020].

G. Tanner, *Creating your own object detector*. Towards Data Science [En línia] Disponible: https://towardsdatascience.com/creating-your-own-object-detector-ad69dda69c85. 2019 [Accedit el 9/5/2020].

G. Tanner, *Live Object Detection*. Towards Data Science [En línia] Disponible: https://towardsdatascience.com/live-object-detection-26cd50cceffd. 2019 [Accedit el 10/5/2020].

I. Mihajlovic, *Everything You Ever Wanted To Know About Computer Vision.*. Toward Data Science [En línia] Disponible: https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e. 2019 [Accedit el 9/4/2020].

J. Gallostra Acín, Kyle, *ArduinoLatexListing*. GitHub, [En línia] Disponible: https://github.com/trihedral/ArduinoLatexListing. [Accedit el 19/6/2020].

J. Le, *Haar Cascade Training on Windows by GUI Tool*. YouTube [En línia] Disponible: https://www.youtube.com/watch?v=_NlaeY9Pp10. 2017 [Accedit el 12/6/2020].

J. Penalva, *46 proyectos makers para hacer en verano con Arduino y Raspberry Pi.* xataka [En línia] Disponible: https://www.xataka.com/makers/46-proyectos-makers-para-hacer-verano-arduino-raspberry-pi. 2018 [Accedit el 14/6/2020].

Linuxize, *How to Install OpenCV on Ubuntu 18.04.* [En línia] Disponible: https://linuxize.com/post/how-to-install-opencv-on-ubuntu-18-04/. [Accedit el 15/6/2020].

M. J. Thompson, *Python: 3 Manuscripts in 1 book: - Python Programming For Beginners - Python Programming For Intermediates - Python Programming for Advanced.* 2018 e-Book.

M. Jones, P. Viola, *Rapid Object Detection using a Boosted Cascade of Simple Features.* [En línia] Disponible: https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf. 2001 [Accedit el 17/6/2020].

MarkDaoust, *Tensorflow Object Detection API.* GitHub [En línia] Disponible: https://github.com/tensorflow/models/tree/master/research/object_detection. [Accedit el 26/5/2020].

M. Zhu, S. Guadarrama, pkulzc, *Models Configuration Files.* GitHub [En línia] Disponible: https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs. [Accedit el 2/6/2020].

OpenCV, *Cascade Classifier.* [En línia] Disponible: https://docs.opencv.org/master/db/d28/tutorial_cascade_classifier.html. [Accedit el 17/6/2020].

OpenCV, *Installation in Linux.* [En línia] Disponible: https://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html. 2011. [Accedit el 28/3/2020].

OpenCV, *Tutorials.* [En línia] Disponible: https://docs.opencv.org/master/d9/df8/tutorial_root.html. [Accedit el 26/2/2020].

Open Images Dataset, *Open Images Dataset V6 + Extensions.* Googleapis [En línia] Disponible: https://storage.googleapis.com/openimages/web/index.html. [Accedit el 14/6/2020].

philferriere, *Clone of COCO API - http://cocodataset.org/.* GitHub [En línia] Disponible: https://github.com/philferriere/cocoapi. [Accedit el 30/5/2020].

Presentitude, *Colors on the color wheel: The 4 important color models for presentation design.* SlideShare [En línia] Disponible: https://www.slideshare.net/Present

itude/the-4-important-color-models-for-presentation-design/16-Colors_on_the_color_wheel. 2016 [Accedit el 10/4/2020].

PyPi, *tensorflow 2.2.0*. [En línia] Disponible: https://pypi.org/project/tensorflow/. [Accedit el 13/6/2020].

Python, *General Python FAQ*. [En línia] Disponible: https://docs.python.org/3/faq/general.html#what-is-python%22. 2001 [Accedit el 9/4/2020].

R. Gour, *Various Uses of TensorFlow*. DZone [En línia] Disponible: https://dzone.com/articles/learn-various-uses-of-tensorflow. 2019 [Accedit el 13/6/2020].

Raspberry Pi, *GPIO*. [En línia] Disponible: https://www.raspberrypi.org/documentation/usage/gpio/. [Accedit el 12/4/2020].

Raspberry Pi, *Raspberry Pi 3 Model B+*. [En línia] Disponible: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. [Accedit el 5/4/2020].

Ray, *Operador Sobel para la detección de bordes en imágenes.*. Programación Extrema [En línia] Disponible: https://programacionextrema.es/2017/09/10/operador-sobel-la-deteccion-bordes-imagenes/. 2017 [Accedit el 11/6/2020]
SAS, *Computer Vision: What it is and why it matters*. [En línia] Disponible: https://www.sas.com/en_us/insights/analytics/computer-vision.html. [Accedit el 10/4/2020].

sentdex, *Creating your own Haar Cascade OpenCV Python Tutorial*. Python Programming [En línia] Disponible: https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/. [Accedit el 15/6/2020].

sentdex, *Introduction and Use - Tensorflow Object Detection API Tutorial*. Python Programming [En línia] Disponible: https://pythonprogramming.net/introduction-use-tensorflow-object-detection-api-tutorial/. [Accedit el 26/5/2020].

sentdex, *Reading game frames in Python with OpenCV - Python Plays GTA V*. Python Programming [En línia] Disponible: https://pythonprogramming.net/game-frames-open-cv-python-plays-gta-v/. [Accedit el 18/5/2020].

Synced, *Introduction to Deep Learning for Graphs and Where It May Be Heading*. Medium [En línia] Disponible: https://medium.com/syncedreview/introduction-to-deep-learning-for-graphs-and-where-it-may-be-heading-75d48

f42a322. 2020 [Accedit el 10/4/2020].

T. Sel, *TensorFlow For Beginners*. Amazon [En línia] Disponible: https://www.amazon.es/gp/product/B0872T7GS1/ref=kinw_myk_ro_title. e-Book.

T. Stalin, *Computer Vision with OpenCV and Python 3: Practical examples workbook*. Amazon [En línia] Disponible: https://www.amazon.es/gp/product/B078QFL1NY/ref=kinw_myk_ro_title. e-Book.

Tecfluïd, *Glossari de termes tècnics*. [En línia] Disponible: https://tecfluid.com/glossari-de-termes-tecnics/?lang=ca. [Accedit el 14/6/2020].

TensorFlow, *All symbols in TensorFlow 2*. [En línia] Disponible: https://www.tensorflow.org/api_docs/python/. [Accedit el 2/4/2020].

TensorFlow, *Case studies*. [En línia] Disponible: https://www.tensorflow.org/about/case-studies. [Accedit el 13/6/2020].

TensorFlow Object Detection API Tutorial, *Training Custom Object Detector*. readthedocs [En línia] Disponible: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html. [Accedit el 30/5/2020].

Y. FM, *Qué es Arduino, cómo funciona y qué puedes hacer con uno*. Xatak [En línia] Disponible: https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno. 2018 [Accedit el 12/4/2020].

W. Xing, *Deep learning diagram*. ResearchGate [En línia] Disponible: https://www.researchgate.net/figure/Deep-learning-diagram_fig5_323784695. 2018 [Accedit el 10/4/2020].

Wikipedia, *Bus en sèrie universal*. [En línia] Disponible: https://ca.wikipedia.org/wiki/Bus_en_s%C3%A8rie_universal. [Accedit el 9/4/2020].

Wikipedia, *Bus (informàtica)*. [En línia] Disponible: Bus(inform\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{a\global\mathchardef\accent@spacefactor\spacefactor}\accent18a\egroup\spacefactor\accent@spacefactortica). [Accedit el 13/4/2020].

Wikipedia, *C++*. [En línia] Disponible: https://ca.wikipedia.org/wiki/C++. [Accedit el 9/4/2020].

Wikipedia, *Camera Serial Interface*. [En línia] Disponible: https://en.wikipedia.org/wiki/Camera_Serial_Interface. [Accedit el 3/6/2020].

Wikipedia, *Computer vision*. [En línia] Disponible: https://en.wikipedia.org/wiki/Computer_vision. [Accedit el 10/4/2020].

Wikipedia, *Dataflow programming*. [En línia] Disponible: https://en.wikipedia.org/wiki/Dataflow_programming. [Accedit el 10/6/2020].

Wikipedia, *Deep learning*. [En línia] Disponible: https://en.wikipedia.org/wiki/Deep_learning. [Accedit el 9/4/2020].

Wikipedia, *Graf (matemàtiques)*. [En línia] Disponible: https://ca.wikipedia.org/wiki/Graf_(matem%C3%A0tiques). [Accedit el 10/4/2020].

Wikipedia, I^2C. [En línia] Disponible: https://ca.wikipedia.org/wiki/I%C2%B2C. [Accedit el 13/4/2020].

Wikipedia, *Interfície de programació d'aplicacions*. [En línia] Disponible: https://ca.wikipedia.org/wiki/Interf%C3%ADcie_de_programaci%C3%B3_d'aplicacions. [Accedit el 12/6/2020].

Wikipedia, *Interfície gràfica d'usuari*. [En línia] Disponible: https://ca.wikipedia.org/wiki/Interf%C3%ADcie_gr%C3%A0fica_d%27usuari. [Accedit el 13/6/2020].

Wikipedia, *Llenguatge de màquina*. [En línia] Disponible: https://ca.wikipedia.org/wiki/Llenguatge_de_m%C3%A0quina. [Accedit el 9/4/2020].

Wikipedia, *OpenCV*. [En línia] Disponible: https://es.wikipedia.org/wiki/OpenCV. [Accedit el 3/6/2020].

Wikipedia, *Pantalla de cristall líquid*. [En línia] Disponible: https://ca.wikipedia.org/wiki/Pantalla_de_cristall_l%C3%ADquid. [Accedit el 14/6/2020].

Wikipedia, *Servomotor de modelismo*. [En línia] Disponible: https://es.wikipedia.org/wiki/Servomotor_de_modelismo. [Accedit el 9/4/2020].

Wikipedia, *TensorFlow*. [En línia] Disponible: https://en.wikipedia.org/wiki/TensorFlow. [Accedit el 10/6/2020].

Wikipedia, *Unidad central de procesamiento*. [En línia] Disponible: https://es.wikipedia.org/wiki/Unidad_central_de_procesamiento. [Accedit el 9/4/2020].

Wikipedia, *Unitat de procés gràfic*. [En línia] Disponible: https://ca.wikipedia.o

rg/wiki/Unitat_de_proc%C3%A9s_gr%C3%A0fic. [Accedit el 9/4/2020].

Wikipedia, *Wi-Fi*. [En línia] Disponible: https://ca.wikipedia.org/wiki/Wi-Fi. [Accedit el 9/4/2020].

Pressupost

Concepte		Quant.	Unit.	Preu per unitat (€/u)	Total (€)
Components del vehicle					
	Vehicle	1	u	64,95	64,95
	Raspberry Pi	1	u	37,44	37,44
	Piles	2	u	16,55	33,10
	Carregador de piles	1	u	8,18	8,18
Components del semàfor					
	300 LEDs	1	u	7,99	7,99
	54m de cablejat	1	u	18,99	18,99
	525 resistències	1	u	10,99	10,99
	ELegoo UNO R3	1	u	9,99	9,99
Altres					
	Cartolines de la carretera	3	u	1,80	5,40
	Cinta blanca	1,95	m	0,21	0,41
	Total del vehicle				**194.44**
Escriptura					
	Estudiant	115	h	8	920,00
Programació					
	Estudiant	55	h	8	440,00
Experiments					
	Estudiant	20	h	8	160,00
Estudi					
	Estudiant	90	h	8	720,00
Muntatge d'elements externs					
	Estudiant	12	h	8	96,00
	Total Projecte				**1.940,00**
Llicencies					
	Windows 10 home	1	u	139,00	139,00
	Total llicencies				**139,00**
	Total				**2.273,44**

Taula 6: Resum de les despeses